Уральский федеральный университет
имени первого Президента России Б.Н.Ельцина
Институт радиоэлектроники и информационных технологий — РТФ

ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В ЭКСПЕРТНЫХ СИСТЕМАХ

Учебное пособие

Министерство образования и науки Российской Федерации
Уральский федеральный университет
имени первого Президента России Б. Н. Ельцина

ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В ЭКСПЕРТНЫХ СИСТЕМАХ

Рекомендовано методическим советом
Уральского федерального университета
в качестве **учебного пособия** для студентов вуза
всех форм обучения направления
27.04.04 — Автоматизация и управление

Екатеринбург
Издательство Уральского университета
2017

УДК 004.891:681.518.2(075.8)
ББК 32.813-022я73
П71

Составители В. А. Морозова, В. И. Паутов

Рецензенты: кафедра общих профессиональных дисциплин Уральского технического института связи и информатики (филиал) ФГБОУ ВО Сибирского государственного университета телекоммуникаций и информатики (заместитель завкафедрой, доц., канд. техн. наук *Н. В. Будылдина*); кафедра физики Уральского государственного горного университета (завкафедрой, д-р физ.-мат. наук *И. Г. Коршунов*)

Научный редактор — доц., канд. техн. наук *В. А. Матвиенко*

П71 **Представление знаний в экспертных системах** : учебное пособие / сост. В. А. Морозова, В. И. Паутов. — Екатеринбург : Изд-во Урал. ун-та, 2017. — 120 с.
ISBN 978-5-7996-2037-0

Учебное пособие посвящено одному из центральных вопросов, возникающих при проектировании экспертных систем (ЭС), — проблеме представления знаний. В учебном пособии рассматриваются состав знаний, необходимых для функционирования ЭС, и основные модели представления знаний; особенности представления знаний в существующих экспертных системах и инструментальных средствах для их разработки.

Библиогр.: 71 назв. Рис. 12. Табл. 1.

УДК 004.891:681.518.2(075.8)
ББК 32.813-022я73

ISBN 978-5-7996-2037-0

ВВЕДЕНИЕ

Экспертные системы (ЭС) — первый продукт, появившийся на информационном рынке, как итог 30-летней изыскательской работы в области искусственного интеллекта (ИИ) [1; 2; 3]. ЭС представляют собой программы для компьютера, которые могут воспроизводить процесс решения проблемы человеком-экспертом. В настоящее время интерес к ЭС со стороны промышленности чрезвычайно возрос, поскольку они способны дать средства, стимулирующие повышение производительности труда и увеличение прибыльности производства [4; 5; 6].

По классическому определению экспертная система — это программный продукт, в котором используется искусственный интеллект в виде типовых логических правил, содержащихся в базе знаний, заполняемой экспертом. Работа системы построена по алгоритму, имитирующему мыслительный процесс.

В начале 80-х гг. XX в. в рамках искусственного интеллекта сформировалось самостоятельное направление — «инженерия знаний», в задачу которого входят разработка, исследование и использование экспертных систем. Термин «инженерия знаний» введен Е. Фейгенбаумом как «привнесе-

ние принципов и инструментария исследований из области искусственного интеллекта в решение трудных прикладных проблем, требующих знаний экспертов» [7]. Огромный интерес к ЭС вызван следующими основными причинами. Во-первых, они ориентированы на решение широкого круга задач в неформализованных областях, т. е. на приложения, которые до недавнего времени считались малодоступными для вычислительной техники. Во-вторых, экспертные системы позволяют специалистам, не имеющим навыков программирования, создавать практически значимые приложения, что резко расширяет сферу использования вычислительной техники. В-третьих, экспертные системы при решении практических задач позволяют получать результаты, сравнимые, а иногда и превосходящие те, которые может получить эксперт-человек. В-четвертых, современные ЭС легко объединяются с традиционными программными системами (системами управления базами данных, пакетами прикладных программ и т. д.) в интегрированные приложения.

При рассмотрении ЭС используются следующие основные понятия [8]: предметная область, проблемная область, данные, знания. Под *предметной областью* будем иметь в виду область экспертизы или знания об этой области. Понятие *проблемная область* включает предметную область и задачи, решаемые в этой области. Термин «проблемная область» будет использоваться в тех случаях, когда необходимо подчеркнуть, что речь идет не только об описании фактов области экспертизы, но и о задачах, решаемых в этой области. Под *данными* будем понимать исходные, промежуточные или окончательные данные о решаемой в текущий момент задаче, т. е. данные — это та информация, которая существует в ходе консультаций. Под *знаниями* будем понимать любую информацию (в том числе и конкретные факты), которая хранится в системе вне зависимости от того, решает система задачу или нет.

Настоящее учебное пособие посвящено одному из центральных вопросов, возникающих при проектировании экспертных систем (ЭС), — проблеме представления знаний.

В области экспертных систем представление знаний означает не что иное, как систематизированную методику описания на машинном уровне того, что знает человек-эксперт, специализирующийся в конкретной предметной области.

Любое общение человека с миром техники предполагает наличие некоторого предварительного знания. Если, например, некто берется за поиск неисправности в цифровой схеме, то это предполагает, что он обладает определенными базовыми знаниями из области электротехники и цифровой техники. Нет необходимости подчеркивать, что компьютер (в чистом виде) никакими предварительными знаниями не обладает, а потому техническая экспертность — набор качеств, лежащих в основе высокого уровня работы людей –специалистов при решении проблем в определенной узкой области, — должна включать и эти предварительные знания.

И наконец, представление предполагает определенную организованность знаний. Представление знаний должно позволить извлекать их в нужной ситуации с помощью относительно несложного и более–менее естественного механизма. Простого перевода информации (знаний) в форму, пригодную для хранения на машинных носителях, здесь явно недостаточно. Для того чтобы можно было достаточно быстро извлекать те элементы знаний, которые наиболее пригодны в конкретной ситуации, база знаний должна обладать достаточно развитыми средствами индексирования и контекстной адресации. Тогда программа, использующая знания, сможет управлять последовательностью применения определенных «элементов» знания, даже не обладая точной информацией о том, как они хранятся.

1. ВОПРОСЫ, РЕШАЕМЫЕ ПРИ ПРЕДСТАВЛЕНИИ ЗНАНИЙ

Первый и основной вопрос, который надо решить при представлении знаний, это вопрос определения состава знаний, т. е. определение того «ЧТО представлять» в ЭС [9]. Важность вопроса «ЧТО представлять» определяется тем, что решение именно этой проблемы обеспечивает адекватное отображение моделируемой сущности в системе. Второй вопрос касается того «КАК представлять» знания. Необходимо отметить, что проблемы «ЧТО представлять» и «КАК представлять» не являются независимыми [7; 8].

Вопрос «КАК представлять» можно разделить на две в значительной степени независимые задачи:

1) как организовать (структурировать) знания,
2) как представить знания в выбранном формализме.

Стремление выделить организацию знаний в самостоятельную задачу вызвано, в частности, тем, что эта задача возникает для любого языка представления, и способы решения этой задачи являются одинаковыми (либо сходными) вне зависимости от используемого формализма.

Итак, в круг вопросов, решаемых при представлении знаний, включаются следующие [7; 8]:

- определение состава представляемых знаний;
- организация знаний;
- представление знаний, т. е. определение модели представления;
- использование выбранного представления.

2. СОСТАВ ЗНАНИЙ ЭКСПЕРТНОЙ СИСТЕМЫ

Рассмотрим вопрос о том, какими (по составу) знаниями должна обладать ЭС для выполнения стоящих перед ней задач.

Состав знаний ЭС определяется следующими факторами [7, 8]:

- проблемной средой (областью);
- архитектурой (структурой) ЭС;
- потребностями и целями пользователей;
- языком общения.

В соответствии с общей схемой статической экспертной системы (см. рис. 2.1) для ее функционирования требуются следующие знания [7; 8]:

- знания о процессе решения задачи (т. е. управляющие знания), используемые интерпретатором (решателем);
- знания о языке общения и способах организации диалога, используемые лингвистическим процессором (диалоговым компонентом);
- знания о способах представления и модификации знаний, используемые компонентом приобретения знаний;

- поддерживающие структурные и управляющие знания, используемые объяснительным компонентом.

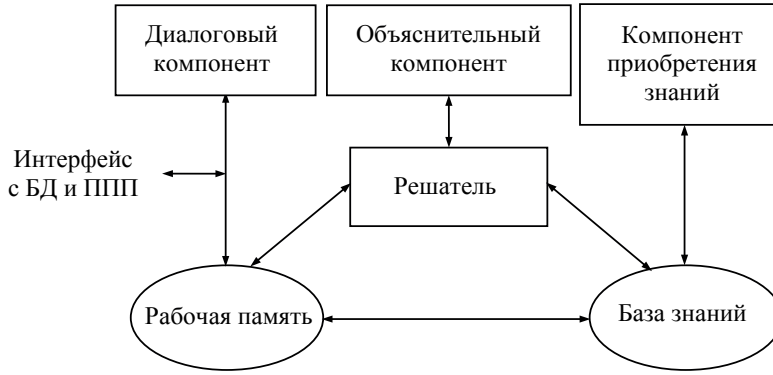


Рис. 2.1. Структура статической ЭС

Для динамической ЭС, кроме того, необходимы следующие знания (см. рис. 2.2):

- знания о методах взаимодействия с внешним окружением;
- знания о модели внешнего мира.

Зависимость состава знаний от требований пользователя проявляется в следующем [7; 8]:

1) какие задачи (из общего набора задач) и с какими данными хочет решать пользователь;
2) каковы предпочтительные способы и методы решения;
3) при каких ограничениях на количество результатов и способ их получения должна быть решена задача;
4) каковы требования к языку общения и к организации диалога;
5) какова степень общности/конкретности знаний о проблемной области, доступная пользователю;
6) каковы цели пользователей.

Рис. 2.2. Архитектура статических и динамических ЭС (компоненты, подвергающиеся изменениям, имеют тень)

Состав знаний о языке общения зависит как от языка общения, так и от требуемого уровня понимания.

Рассмотрим классификацию знаний с точки зрения проблемной области и с точки зрения архитектуры ЭС.

2.1. Классификация знаний с точки зрения проблемной области

Понятие «проблемная среда» включает предметную область (множество сущностей, описывающих область экспертизы, т. е. множество объектов, значений их характеристик и связывающих их отношений) и решаемые в предметной области задачи [7]. Иначе говоря, проблемная среда включает сущности (структуры данных) и решаемые над ними задачи, представляемые в виде исполняемых утверждений (в виде правил, процедур, формул и т. п.). В связи с этим проблемная среда определяется характеристиками соответствующей предметной области и характеристиками типов решаемых в ней задач. Заметим, что наряду с понятием «проблемная среда» используется синонимичный ему термин «проблемная область».

Характеристики предметной области определяются следующим набором параметров [7]:

1) тип предметной области:
- статический, т. е. входные данные не изменяются за время сеанса работы приложения, значения других (не входных) данных изменяются только ЭС;
- динамический, т. е. входные данные, поступающие от внешних источников, изменяются во времени, значения других данных изменяются ЭС или подсистемой моделирования внешнего окружения;

2) способ описания сущностей предметной области:
- совокупность атрибутов и их значений (фиксированный состав сущностей);
- совокупность классов (объектов) и их экземпляров (изменяемый состав сущностей);

3) способ организации сущностей в БЗ:
- неструктурированная БЗ;
- структурирование сущностей БЗ по различным ие-

рархиям (наиболее распространенные иерархии: «общее/частное», «часть/целое»), что обеспечивает наследование свойств сущностей, представляемых в БЗ.

Характеристиками типов задач являются [7]:

1) тип решаемых задач:
- задачи анализа и (или) синтеза;
- статические или динамические задачи;

2) частность (общность) исполняемых утверждений (правил, процедур, формул и т. д.):
- частные (специализированные, конкретные) исполняемые утверждения;
- общие исполняемые утверждения.

Наиболее естественным для человека способом *описания сущностей предметной области* является соотнесение с ними в памяти ЭВМ объектов, состоящих из атрибутов со значениями. Обычно вводится описание объекта некоторого типа, в соответствии с которым создаются конкретные экземпляры объектов этого типа. При этом количество экземпляров объекта никак не ограничивается, т. е. состав представляемых сущностей при таком представлении проблемной области является *изменяемым*.

Однако для простых приложений при малом количестве объектов (от 1 до 5) иногда применяют упрощенное представление в виде атрибут/значение без упоминания объекта, которому эти атрибуты принадлежат. Следствием этого упрощения явилось то, что реально существующие объекты (сущности) предметной области стали представляться в виде фиксированного количества размноженных имен соответствующих атрибутов. Например, вместо ссылки на атрибут $i(1 \div K)$ объекта $X(1 \div N)$ использовали ссылку на атрибут $j(1 \div (K \times N))$. Таким образом, вместо K атрибутов вводили $K \times N$ атрибутов. Достоинства этого подхода состоят в том, что обеспечивается прямая ссылка на атрибут (объект). Недостатки в общем случае весьма значительны. Они заключаются в следующем:

1) невозможно использовать «общие» исполняемые утверждения (правила, процедуры, функции и т. п.), ссылающиеся на произвольное количество сущностей; приходится использовать частные (специализированные) утверждения, что увеличивает их количество. Кроме того, при изменении состава сущностей в БЗ приходится вводить новые соответствующие им специализированные утверждения;
2) устранение объектов не позволяет явно определить естественные взаимосвязи между атрибутами одного объекта (все атрибуты являются как бы независимыми);
3) с устранением объектов исчезает возможность определить иерархию классов объектов [7].

Следующим параметром, характеризующим предметную область, является *наличие (отсутствие) структурирования БЗ*. Смысл структурирования БЗ состоит в следующем:

1) ограничить круг сущностей, которые должны рассматриваться механизмом вывода, и таким образом сократить перебор при выборе решения;
2) обеспечить *наследование свойств сущностей*, т. е. передачу свойств вышестоящих в иерархии сущностей нижестоящим, что значительно упрощает процесс приобретения и использования знаний. Например, общие свойства класса «автомобиль» автоматически наследуются всеми подклассами автомобилей и конкретными экземплярами этих подклассов. В подобной иерархии наследуется отношение «являться подмножеством (экземпляром)». Кроме того, широко применяется и другая иерархия — «является частью».

По типу решаемых задач в первую очередь все задачи целесообразно разделить на задачи анализа и синтеза [7; 8].

В *задаче анализа* задана модель сущности (объекта) и требуется в результате анализа этой модели определить некото-

рые неизвестные характеристики (функции) модели. В задаче синтеза задаются условия, которым должны удовлетворять характеристики (функции) некоторой «неизвестной» модели сущности, и требуется построить модель этой сущности. Решение задачи синтеза представляет собой итерационный процесс, состоящий из следующих шагов [7]:

1) создание исследовательской модели сущности;
2) анализ этой модели (т. е. решение задачи анализа);
3) сравнение результатов анализа с условиями задачи.

Таким образом, задача синтеза включает в себя анализ. Отметим, что в процессе создания конкретной ЭС, решающей задачу анализа, разработчик, создавая БЗ (модель области экспертизы), решает задачу синтеза, а построенная ЭС будет решать задачу анализа.

Задачи синтеза и анализа могут решаться в статических и динамических областях. Если ЭС базируется на предположении, что исходная информация о предметной области (об окружающем мире), на основе которой решается задача, не изменяется за время решения задач, то говорят о статической предметной области (точнее о статическом представлении области в ЭС); если информация о предметной области изменяется за время решения задач, то говорят о *динамической предметной области*. При представлении динамической области возникает задача моделирования окружающего мира, в частности моделирования активных агентов.

Если задачи, решаемые ЭС, явно не учитывают фактор времени и (или) не изменяют в процессе своего решения данные (знания) об окружающем мире, то говорят, что ЭС решает *статические задачи*; если задачи учитывают фактор времени и (или) изменяют в процессе решения данные об окружающем мире, то говорят о решении *динамических задач*. Таким образом, ЭС работает в статической проблемной среде, если она использует статическое представление и решает статиче-

ские задачи. ЭС работает в динамической проблемной среде, если она использует динамическое представление и (или) решает динамические задачи.

Учитывая значимость времени в динамических проблемных средах, многие специалисты называют их приложениями (ЭС), работающими в реальном времени. Обычно выделяют следующие системы реального времени: «псевдореального» времени, «мягкого» реального времени и «жесткого» реального времени. Системы псевдореального времени, как следует из названия, не являются системами реального времени, однако они в отличие от статических систем получают и обрабатывают данные, поступающие от внешних источников. Системы псевдореального времени решают задачу быстрее, чем происходят значимые изменения информации об окружающем мире.

Системы «мягкого» реального времени работают в тех приложениях, где допустимо время реакции на события более 0,1–1 с. К этому диапазону относятся почти все существующие ЭС реального времени. Системы «жесткого» реального времени должны обеспечивать время реакции быстрее 0,1–0,5 с. Для достижения такого быстродействия они используют не стандартные операционные системы (ОС) типа Unix и Windows NT, а специализированные ОС и специализированные бортовые ЭВМ, обеспечивающие быстрое время реакции. В настоящее время ЭС, работающие в «жестком» реальном времени, нам не известны.

Задачи, решаемые ЭС, различают тем, как представляются исполняемые утверждения. Используются как *частные (специализированные) утверждения*, т. е. утверждения, содержащие ссылки на конкретные сущности (объекты), так и *общие утверждения*, относящиеся к любым сущностям заданного типа (вне зависимости от их числа и имени). Использование общих утверждений позволяет значительно лаконичнее представлять знания. Однако так как общие утверждения

не содержат явных ссылок на конкретные сущности, для их использования требуется затратить значительную работу по определению тех сущностей, к которым они должны применяться, т. е. выполнить, как говорят специалисты, операцию сопоставления [7, гл. 6].

Не все сочетания перечисленных выше параметров, характеризующих проблемную среду, встречаются на практике. Выделим несколько наиболее часто встречающихся типов проблемных сред.

Тип 1. Статическая проблемная среда: статическая предметная область; сущности представляются как совокупность атрибутов и их значений; состав сущностей неизменяемый; БЗ не структурирована; решаются статические задачи анализа, используются только специализированные исполняемые утверждения.

Тип 2. Статическая проблемная среда: статическая предметная область; сущности представляются в виде атрибутов со значениями или вырожденных объектов (фреймов); состав сущностей неизменяемый; иерархия БЗ либо отсутствует, либо слабо выражена (нет наследования свойств); решаются статические задачи анализа, используются специализированные исполняемые утверждения.

Тип 3. Статическая проблемная среда: статическая предметная область; сущности представляются в виде объектов; состав сущностей изменяемый; БЗ структурирована; решаются статические задачи анализа и синтеза, используются общие и специализированные исполняемые утверждения.

Тип 4. Динамическая проблемная среда: динамическая предметная область; сущности представляются совокупностью атрибутов и их значений; состав сущностей неизменяемый; БЗ не структурирована; решаются динамические задачи анализа, используются специализированные исполняемые утверждения.

Тип 5. Динамическая проблемная среда: динамическая предметная область; сущности представляются в виде объектов; изменяемый состав сущностей; БЗ структурирована; решаются динамические задачи анализа и синтеза; используются общие и специализированные исполняемые утверждения.

2.2. Классификация знаний с точки зрения архитектуры ЭС

С точки зрения архитектуры ЭС знания могут быть структурированы так, как показано на рис. 2.3. Данная классификация рассматривает знания, учитывая их использование в ходе работы экспертной системы. С учетом архитектуры экспертной системы знания целесообразно делить на *интерпретируемые* и *неинтерпретируемые* [7; 8]. К первому типу относятся те знания, которые способен интерпретировать решатель (интерпретатор). Все остальные знания относятся ко второму типу. Решатель не знает их структуры и содержания. Если эти знания используются каким-либо компонентом системы, то он не «осознает» этих знаний.

Неинтерпретируемые знания подразделяются на *вспомогательные*, хранящие информацию о лексике и грамматике языка общения, информацию о структуре диалога, и *поддерживающие* знания. Вспомогательные знания обрабатываются естественно-языковой компонентой, но ход этой обработки решатель не осознает, так как этот этап обработки входных сообщений является вспомогательным для проведения экспертизы. Поддерживающие знания используются при создании системы и при выполнении объяснений. Поддерживающие знания выполняют роль описаний (обоснований) как интерпретируемых знаний, так и действий системы. Под-

держивающие знания подразделяются на *технологические* и *семантические*. Технологические поддерживающие знания содержат сведения о времени создания описываемых ими знаний, об авторе знаний и т. п. Семантические поддерживающие знания содержат смысловое описание этих знаний. Они содержат информацию о причинах ввода знаний, о назначении знаний, описывают способ использования знаний и получаемый эффект. Поддерживающие знания имеют описательный характер.

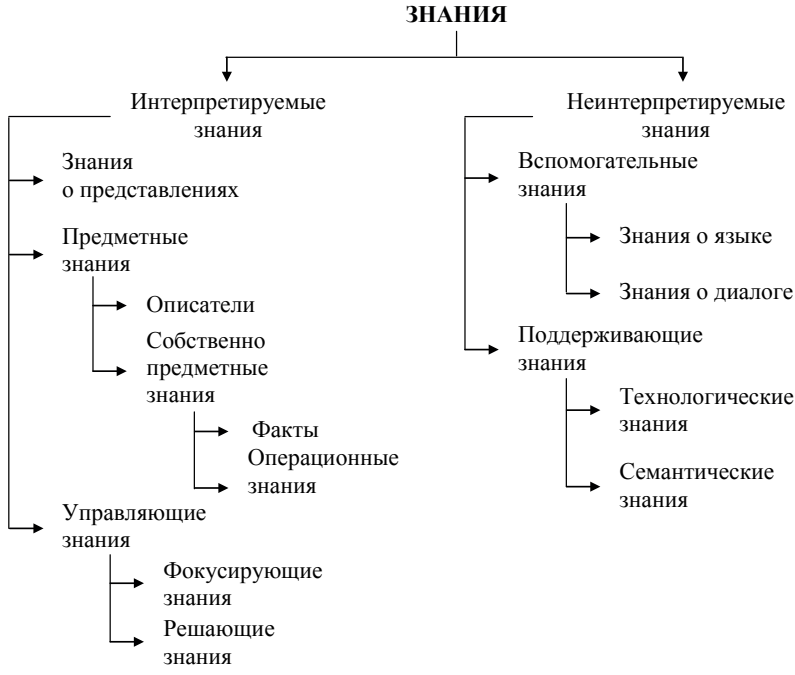


Рис. 2.3. Структура знаний в базе знаний

Интерпретируемые знания можно разделить на *предметные знания*, *управляющие знания* и *знания о представлении*. Знания о представлении содержат информацию о том, каким образом (в каких структурах) в системе представлены интерпретируемые знания.

Предметные знания содержат данные о предметной области и способах преобразования этих данных при решении поставленных задач. Отметим, что по отношению к предметным знаниям знания о представлении и знания об управлении являются метазнаниями. В предметных знаниях можно выделить описатели и собственно предметные знания. Описатели содержат определенную информацию о предметных знаниях, такую как коэффициент определенности правил и данных, меры важности и сложности. Собственно предметные знания разбиваются на *факты* и *исполняемые утверждения*. Факты определяют возможные значения сущностей и характеристик предметной области. Исполняемые утверждения содержат информацию о том, как можно изменять описание предметной области в ходе решения задач.

Говоря другими словами, исполняемые *утверждения* — это знания, задающие процедуры обработки. Однако использовать термин «процедурные знания» не рекомендуется, так как эти знания могут быть заданы не только в процедурной, но и в декларативной форме.

Управляющие знания можно разделить на *фокусирующие* и *решающие*. Фокусирующие знания описывают, какие знания следует использовать в той или иной ситуации. Обычно фокусирующие знания содержат сведения о наиболее перспективных объектах или правилах, которые целесообразно использовать при проверке соответствующих гипотез [7]. В первом случае внимание фокусируется на элементах рабочей памяти, во втором — на правилах базы знаний. Решающие знания содержат информацию, используемую для выбора способа ин-

терпретации знаний, подходящего к текущей ситуации. Эти знания применяются для выбора стратегий или эвристик, наиболее эффективных для решения данной задачи.

2.3. Использование метазнаний

Качественные и количественные показатели экспертной системы могут быть значительно улучшены за счет использования метазнаний, т. е. знаний о знаниях. Метазнания не представляют некоторую единую сущность, они могут применяться для достижения различных целей [7; 8].

Перечислим возможные назначения метазнаний [8, гл. 6]:

1) метазнания в виде стратегических метаправил используются для выбора релевантных правил;
2) метазнания используются для обоснования целесообразности применения правил из области экспертизы;
3) метаправила используются для обнаружения синтаксических и семантических ошибок в предметных правилах;
4) метаправила позволяют системе адаптироваться к окружению путем перестройки предметных правил и функций;
5) метаправила позволяют явно указать возможности и ограничения системы, т. е. определить, что система знает, а что не знает.

3. ОРГАНИЗАЦИЯ ЗНАНИЙ

Вопросы организации знаний необходимо рассматривать в любом представлении, и их решение в значительной степени не зависит от выбранного способа (модели) представления. Выделим следующие аспекты проблемы организации знаний [7; 8]:

- организация знаний по уровням представления и по уровням детальности;
- организация знаний в рабочей памяти экспертной системы;
- организация знаний в базе знаний.

3.1. Уровни представления и уровни детальности

Для того чтобы экспертная система могла управлять процессом поиска решения, была способна приобретать новые знания и объяснять свои действия, она должна уметь не только использовать свои знания, но и обладать способностью понимать и исследовать их, т. е. экспертная система должна

иметь знания о том, как представлены ее знания о проблемной среде [7; 8]. Если знания о проблемной среде назвать знаниями нулевого уровня представления, то первый уровень представления содержит метазнания, т. е. знания о том, как представлены во внутреннем мире системы знания нулевого уровня. Первый уровень содержит знания о том, какие средства используются для представления знаний нулевого уровня. Знания первого уровня играют существенную роль при управлении процессом решения, при приобретении и объяснении действий системы. В связи с тем, что знания первого уровня не содержат ссылок на знания нулевого уровня, знания первого уровня независимы от проблемной среды.

Число уровней представления может быть больше двух. Второй уровень представления содержит сведения о знаниях первого уровня, т. е. знания о представлении базовых понятий первого уровня. Разделение знаний по уровням представления обеспечивает расширение области применимости системы [7; 8].

Выделение уровней детальности позволяет рассматривать знания с различной степенью подробности. Количество уровней детальности во многом определяется спецификой решаемых задач, объемом знаний и способом их представления. Как правило, выделяется не менее трех уровней детальности, отражающих соответственно общую, логическую и физическую организацию знаний. Введение нескольких уровней детальности обеспечивает дополнительную степень гибкости системы, так как позволяет производить изменения на одном уровне, не затрагивая другие. Изменения на одном уровне детальности могут приводить к дополнительным изменениям на этом же уровне, что оказывается необходимым для обеспечения согласованности структур данных и программ. Однако наличие различных уровней препятствует распространению изменений с одного уровня на другие [7; 8].

3.2. Организация знаний в рабочей памяти

Рабочая память (РП) экспертных систем предназначена для хранения данных. Данные в рабочей памяти могут быть однородны или разделяются на уровни по типам данных. В последнем случае на каждом уровне рабочей памяти хранятся данные соответствующего типа. Выделение уровней усложняет структуру экспертной системы, но делает систему более эффективной. Например, можно выделить уровень планов, уровень агенды (упорядоченного списка правил, готовых к выполнению) и уровень данных предметной области (уровень решений) [7; 8].

В современных экспертных системах данные в рабочей памяти рассматриваются как изолированные или как связанные. В первом случае рабочая память состоит из множества простых элементов, а во втором — из одного или нескольких (при нескольких уровнях в РП) сложных элементов (например, объектов). При этом сложный элемент соответствует множеству простых, объединенных в единую сущность. Теоретически оба подхода обеспечивают полноту, но использование изолированных элементов в сложных предметных областях приводит к потере эффективности [7; 8].

Данные в РП в простейшем случае являются *константами* и (или) *переменными*. При этом переменные могут трактоваться как характеристики некоторого объекта, а константы — как значения соответствующих характеристик. Если в РП требуется анализировать одновременно несколько различных объектов, описывающих текущую проблемную ситуацию, то необходимо указывать, к каким объектам относятся рассматриваемые характеристики. Одним из способов решения этой задачи является явное указание того, к какому объекту относится характеристика [7; 8].

Если РП состоит из сложных элементов, то связь между отдельными объектами указывается явно, например заданием семантических отношений. При этом каждый объект может иметь свою внутреннюю структуру. Необходимо отметить, что для ускорения поиска и сопоставления данные в РП могут быть связаны не только логически, но и ассоциативно [7; 8].

3.3. Организация знаний в базе знаний

Показателем интеллектуальности системы с точки зрения представления знаний считается способность системы использовать в нужный момент необходимые (релевантные) знания [9]. Системы, не имеющие средств определения релевантных знаний, неизбежно сталкиваются с проблемой «комбинаторного взрыва». Можно утверждать, что эта проблема является одной из основных причин, ограничивающих сферу применения экспертных систем. В проблеме доступа к знаниям можно выделить три аспекта: связность знаний и данных; механизм доступа к знаниям; способ сопоставления [7; 8; 9].

Связность (агрегация) знаний является основным способом, обеспечивающим ускорение поиска релевантных знаний. Большинство специалистов пришли к убеждению, что знания следует организовывать вокруг наиболее важных объектов (сущностей) предметной области. Все знания, характеризующие некоторую сущность, связываются и представляются в виде отдельного объекта. При подобной организации знаний, если системе потребовалась информация о некоторой сущности, то она ищет объект, описывающий эту сущность, а затем уже внутри объекта отыскивает информацию о данной сущности. В объектах целесообразно выделять два типа

связок между элементами: *внешние* и *внутренние*. Внутренние связки объединяют элементы в единый объект и предназначены для выражения структуры объекта. Внешние связки отражают взаимозависимости, существующие между объектами в области экспертизы. Многие исследователи классифицируют внешние связки на *логические* и *ассоциативные*. Логические связки выражают семантические отношения между элементами знаний. Ассоциативные связки предназначены для обеспечения взаимосвязей, способствующих ускорению процесса поиска релевантных знаний [7; 8].

Основной проблемой при работе с большой базой знаний является проблема поиска знаний, релевантных решаемой задаче. В связи с тем, что в обрабатываемых данных может не содержаться явных указаний на значения, требуемые для их обработки, необходим более общий механизм доступа, чем метод прямого доступа (метод явных ссылок). Задача этого механизма состоит в том, чтобы по некоторому описанию сущности, имеющемуся в рабочей памяти, найти в базе знаний объекты, удовлетворяющие этому описанию. Очевидно, что упорядочение и структурирование знаний могут значительно ускорить процесс поиска [7; 8].

Нахождение желаемых объектов, в общем случае, уместно рассматривать как двухэтапный процесс. На первом этапе, соответствующем процессу выбора по ассоциативным связкам, совершается предварительный выбор в базе знаний потенциальных кандидатов на роль желаемых объектов. На втором этапе путем выполнения операции сопоставления потенциальных кандидатов с описаниями кандидатов осуществляется окончательный выбор искомых объектов. При организации подобного механизма доступа возникают определенные трудности: Как выбрать критерий пригодности кандидата? Как организовать работу в конфликтных ситуациях? и т. п. [7; 8].

Операция сопоставления может использоваться не только как средство выбора нужного объекта из множества кандидатов; она может быть использована для классификации, подтверждения, декомпозиции и коррекции. Для идентификации неизвестного объекта он может быть сопоставлен с некоторыми известными образцами. Это позволит классифицировать неизвестный объект как такой известный образец, при сопоставлении с которым были получены лучшие результаты. При поиске сопоставление используется для подтверждения некоторых кандидатов из множества возможных. Если осуществлять сопоставление некоторого известного объекта с неизвестным описанием, то в случае успешного сопоставления будет осуществлена частичная декомпозиция описания [7; 8].

Операции сопоставления весьма разнообразны. Обычно выделяют следующие их формы: синтаксическое, параметрическое, семантическое и принуждаемое сопоставления [7; 8; 9]. В случае *синтаксического сопоставления* соотносят формы (образцы), а не содержание объектов. Успешным является сопоставление, в результате которого образцы оказываются идентичными. Обычно считается, что переменная одного образца может быть идентична любой константе (или выражению) другого образца. Иногда на переменные, входящие в образец, накладывают требования, определяющие тип констант, с которыми они могут сопоставляться. Результат синтаксического сопоставления является бинарным: образцы сопоставляются или не сопоставляются. В *параметрическом сопоставлении* вводится параметр, определяющий степень сопоставления. В случае *семантического сопоставления* соотносятся не образцы объектов, а их функции. В случае *принуждаемого сопоставления* один сопоставляемый образец рассматривается с точки зрения другого. В отличие от других типов сопоставления здесь всегда может быть получен поло-

жительный результат. Вопрос состоит в силе принуждения. Принуждение могут выполнять специальные процедуры, связываемые с объектами. Если эти процедуры не в состоянии осуществить сопоставление, то система сообщает, что успех может быть достигнут только в том случае, если определенные части рассматриваемых сущностей можно считать сопоставляющимися [7; 8].

4. МОДЕЛИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ

Перед тем как перейти к описанию моделей представления, выскажем несколько общих соображений [7; 8].

Вопрос выбора модели представления знания сводят к обсуждению баланса между декларативным представлением (ДП) и процедурным представлением (ПП). Различие между ДП и ПП можно выразить как различие между «ЗНАТЬ ЧТО» и «ЗНАТЬ КАК». ПП основано на предпосылке, что интеллектуальная деятельность есть значение проблемной области, вложенное в программы, т. е. знание о том, как можно использовать те или иные сущности. ДП основано на предпосылке, что знание неких сущностей («ЗНАТЬ ЧТО») не имеет глубоких связей с процедурами, используемыми для обработки этих сущностей. При использовании ДП считается, что интеллектуальность базируется на некотором универсальном множестве процедур, обрабатывающих факты любого типа, и на множестве специфических фактов, описывающих частную область знаний [9]. Основное достоинство ДП по сравнению с ПП заключается в том, что в ДП нет необходимости указывать способ использования конкретных фрагментов знания. Простые

утверждения могут использоваться несколькими способами, и может оказаться неудобным фиксировать эти способы заранее. Указанное свойство обеспечивает гибкость и экономичность ДП. Позволяет по-разному использовать одни и те же факты [7; 8].

В ДП знание рассматривается как множество независимых или слабо зависимых фактов, что позволяет осуществлять модификацию знаний и обучение простым добавлением или устранением утверждений. Для ПП проблема модификации значительно сложнее, т. к. здесь необходимо учитывать, каким образом используется данное утверждение. Однако известно, что существует значительное количество сущностей, которые удобно представить в виде процедур и весьма трудно — в чисто декларативном представлении. Желание использовать достоинства ДП и ПП привело к разработке формализмов, использующих смешанное представление, т. е. декларативное представление с присоединенными процедурами (например, фрейм-представление или сети с присоединенными процедурами) или процедурное представление в виде модулей с декларативными образцами. В наиболее совершенном виде эта проблема реализована в объектно-ориентированном подходе [7, п. 5.3].

Модели представления знаний обычно делят на *логические* (формальные) и *эвристические* (формализованные) [7; 8]. В основе *логических моделей* представления знаний лежит понятие формальной системы (теории). Примерами формальных теорий могут служить исчисление предикатов [10; 11] и любая конкретная система продукций [12]. В логических моделях, как правило, используется исчисление предикатов первого порядка, дополненное рядом эвристических стратегий. Особенно активизировалось использование исчисления предикатов после создания мощных процедур поиска вывода, а именно метода резолюций [11] и обратного метода [13]. Эти

методы были обогащены эвристическими процедурами, которые существенно повысили эффективность вывода [11; 14]. Эти методы являются системами *дедуктивного типа*, т. е. в них используется модель получения вывода из заданной системы посылок с помощью фиксированной системы правил вывода. Дальнейшим развитием предикатных систем являются системы *индуктивного типа*, в которых правила вывода порождаются системой на основе обработки конечного числа обучающих примеров [15]. Особое место среди логик, используемых для представления знаний, занимают логики отношений, получивших название *псевдофизических* [16; 17; 18]. Особенность этих логик состоит в использовании в правилах вывода конкретных знаний о свойствах отношений внутри предметных областей.

В логических моделях представления знаний отношения, существующие между отдельными единицами знаний, выражаются только с помощью тех небогатых средств, которые предоставляются синтаксическими правилами используемой формальной системы.

В отличие от формальных моделей *эвристические модели* имеют разнообразный набор средств, передающих специфические особенности той или иной проблемной области. Именно поэтому эвристические модели превосходят логические как по возможности адекватно представить проблемную среду, так и по эффективности используемых правил вывода. К эвристическим моделям, используемым в экспертных системах, можно отнести *сетевые*, *фреймовые*, *продукционные* и *объектно-ориентированные* модели. Следует отметить, что продукционные модели, используемые для представления знаний в экспертных системах, отличаются от формальных продукционных систем [7, п. 1.2]; [12] тем, что они используют более сложные конструкции правил, а также содержат эвристическую информацию о специфике

проблемной среды, выражаемую часто в виде семантических структур.

4.1. Логическая модель представления знаний

В основе логических моделей лежит понятие формальной теории, задаваемой четверкой [7; 8; 9; 19],

$$S = < B, F, A, R >,$$

где B — счетное множество *базовых символов (алфавит)* теории S;

F — подмножество выражений теории S, называемых *формулами теории* (под выражениями понимаются конечные последовательности базовых символов теории S). Обычно существует эффективная процедура (множество синтаксических правил), позволяющая строить из B синтаксически правильные выражения — формулы;

A — выделенное множество формул, называемых *аксиомами теории S*, т. е. множество априорно истинных формул;

R — конечное множество отношений $\{r_1, \ldots, r_n\}$ между формулами, называемыми *правилами вывода*.

Для каждого r_i существует целое положительное число j, такое, что для каждого множества, состоящего из j формул, и для каждой формулы f эффективно решается вопрос о том, находятся ли данные j формул в отношении r_i с формулой f. Если отношение r_i выполняется, то f называется *непосредственным следствием* данных j формул по правилу r_i. *Следствием (выводом)* формулы f_n в теории S называется любая последовательность $f_1, \ldots, f_n$ формул, такая, что для любого i формула f_i есть либо аксиома теории S, либо непосредственное следствие каких-либо предыдущих формул по одному из правил выво-

да. Правила вывода позволяют расширять множество формул, которые считаются истинными в рамках данной теории [7; 8].

Формальная теория называется *разрешимой*, если существует единая эффективная процедура, позволяющая узнать для любой данной формулы, существует ли ее вывод в S. Формальная система S называется *непротиворечивой*, если не существует формулы A, такой, что A и $\neg A$ выводимы в S [7; 8].

Наиболее распространенной формальной системой, используемой для представления знаний, является исчисление предикатов первого порядка. Алфавит исчисления предикатов состоит из следующего набора символов [7; 8]:

- знаков пунктуации {, (, , ,), . , ;};
- пропозициональных связок $\{\neg, \Lambda, V, \supset\}$;
- знаков-кванторов $\{\forall, \exists\}$;
- символов переменных x_k, $k = 1, 2, \ldots$;
- n-местных функциональных букв: f_k^n, $k \geqslant 1$, $n \geqslant 0$ (f_{k0} называют константными буквами);
- n-местных предикатных букв (символов): p_k^n, $k \geqslant 1$, $n \geqslant 1$.

В дальнейшем для упрощения вместо x_k будем употреблять $u, v, x, y, z, \ldots$; вместо f_k^0 — $a, b, c, d, \ldots$; вместо f_k^n $(n \neq 0)$ — $f, g, h, \ldots$; а вместо p_k^n — P, Q, R, S, T, V, W.

Из символов алфавита можно строить различные выражения. Выделяют термы, элементарные формулы (атомы) и правильно построенные формулы (или просто формулы). Всякий символ переменной или константной буквы есть терм. Если $t_1, \ldots, t_n$ $(n \geqslant 1)$ — термы, то и $f_k^n(t_1, \ldots, t_n)$ является термом [7; 8].

Если p_k^n — предикатная буква, а $t_1, \ldots, t_n$ — термы, то $p_k^n(t_1, \ldots, t_n)$ — *элементарная формула* (атом). Атом является *правильно построенной формулой*. Если A и B — правильно построенные формулы, то $\neg A, A \vee B, A \wedge B, A \supset B$ есть правильно построенные формулы. Если A — правильно построенная формула и x — переменная в A, то конструкции $(\forall x)A$ и $(\exists x)A$ —

правильно построенные формулы. Выражение является правильно построенной формулой, только если оно получено с соблюдением приведенных выше правил [7; 8].

Для того чтобы придать формуле содержание, ее интерпретируют как утверждение, касающееся рассматриваемой предметной области. Под *интерпретацией* понимают всякую систему, состоящую из непустого множества D, называемого *областью интерпретации*, и какого-либо соответствия, относящего каждой предикатной букве p_k^n некоторое n-местное отношение в D; каждой функциональной букве f_k^n — некоторую n-местную функцию, отображающую $D^n \to D$, и каждой константной букве f_k^0 — некоторый элемент из D. При заданной интерпретации переменные мыслятся «пробегающими» все значения в области D этой интерпретации, а всякой элементарной формуле приписывается значение «истинно» (И) или «ложно» (Л). Приписывание значения элементарной формуле $p_k^n(t_1, \ldots, t_n)$ осуществляется по следующему правилу: если термы предикатной буквы соответствуют элементам из D, удовлетворяющим отношению, определяемому данной интерпретацией, то значением элементарной формулы будет истина, в противном случае — ложь. Значение неэлементарной формулы вычисляется рекуррентно, исходя из значений составляющих ее формул. Очевидно, что значения формул могут быть истинными или ложными в зависимости от выбранной интерпретации [7; 8].

Основной задачей, решаемой в рамках исчисления предикатов, является выяснение истинности или ложности заданной формулы на некоторой области интерпретации. При этом особая роль отводится *общезначимым* формулам, т. е. формулам, истинным при любой интерпретации, и невыполнимым формулам, т. е. формулам, ложным при любой интерпретации. Справедлива следующая основополагающая *теорема дедукции* [10]: «Пусть даны формулы $B_1, \ldots, B_n$ и формула A.

Формула A является *логическим следствием* $B_1, \ldots, B_n$ тогда и только тогда, когда формула $B_1 \wedge \ldots \wedge B_n \supset A$ общезначима, т. е. $\models (B_1 \wedge \ldots \wedge B_n) \supset A$». Напомним, что формула A логически следует из формул $B_1, \ldots, B_n$ тогда и только тогда, когда всякая интерпретация I, удовлетворяющая $B_1 \wedge \ldots \wedge B_n$, удовлетворяет также и A. Формулы $B_1, \ldots, B_n$ называют *посылками*, а A — *заключением логического следования* и обозначают: $B_1, \ldots, B_n \models A$.

Задачей доказательства теоремы называют выяснение вопроса логического следования некоторой формулы A из заданного множества формул $B_1, \ldots, B_n$, что равносильно доказательству общезначимости формулы $B_1 \wedge \ldots \wedge B_n \supset A$ или невыполнимости формулы $B_1 \wedge \ldots \wedge B_n \wedge \neg A$.

Известно, что для исчисления предикатов первого порядка не существует общего метода установления общей значимости любых формул, т. е. исчисление предикатов первого порядка является неразрешимым. Однако если некоторая формула исчисления предикатов общезначима, то существует процедура для проверки ее общезначимости, т. е. исчисление предикатов можно назвать *полуразрешимым*. Наиболее известными методами доказательства теорем являются метод резолюции [7, гл. 5] и обратный метод [13].

Приведем пример записи некоторого факта в виде формулы исчисления предикатов [7]:

ДАТЬ (МИХАИЛ, ВЛАДИМИРУ, КНИГУ);

$(\exists x)$ (ЭЛЕМЕНТ (x, СОБЫТИЕ — ДАТЬ) $\wedge$

ИСТОЧНИК (x, МИХАИЛ) $\wedge$ АДРЕСАТ (x, ВЛАДИМИР) $\wedge$

ОБЪЕКТ (x, КНИГА)).

Здесь описаны два способа записи одного факта; «Михаил дал книгу Владимиру».

Основным достоинством использования исчисления предикатов в качестве модели представления знаний является наличие формальной единообразной процедуры доказательства теорем. Однако высокая степень единообразия влечет

за собой и основной недостаток данного подхода — сложность использования при доказательстве эвристик, отражающих специфику конкретной проблемной среды. Указанный недостаток особенно важен при построении экспертных систем, вычислительная мощность которых в основном определяется знаниями, характеризующими специфику проблемной среды. К другим недостаткам формальных систем следует отнести их *монотонность* [7, п. 6.3.3] и [8], отсутствие средств для структурирования используемых элементов и недопустимость противоречий [7, гл. 5].

Стремление устранить недостатки формальных систем при их использовании в качестве моделей представления привело к появлению семиотических систем [7; 8; 17; 19; 20]. Семиотическая система формально задается восьмеркой:

$$S = <B, F, A, R, Q\,(B), Q\,(F), Q\,(A), Q\,(R)>.$$

Здесь первые четыре компонента те же, что и в определении формальной системы (см. выше), а остальные компоненты — правила изменения первых четырех компонентов под влиянием накапливаемого в базе знаний интеллектуальной системы опыта о строении и функционировании сущностей в данной проблемной среде (области). Необходимо отметить явную близость понятий «семиотическая система» и «системы возможных миров» Крипке [21; 22; 23]. Можно считать, что правила $Q\,(B)$, $Q\,(F)$, $Q\,(A)$, $Q\,(R)$ после каждого своего применения фиксируют некоторый возможный мир, полностью описываемый соответствующей формальной моделью. При такой трактовке семиотическая система описывается на двух уровнях: на нижнем уровне модель представления знаний описывается традиционной формальной системой, а на верхнем уровне задается модель перестройки (адаптации) этой формальной системы [20]. Теория таких систем находится на начальной стадии развития.

4.2. Продукционная модель

Продукционная модель, или модель, основанная на *правилах*, позволяет представить знания в виде предложений типа «*Если (условие), то (действие)*», где *условие* — это образец, по которому осуществляется поиск в базе; *действие* — действия или операторы, выполняемые при успешном исходе поиска (могут быть промежуточными, выступающими далее как условия, и терминальными или целевыми, завершающими работу системы) [24].

При использовании продукционной модели база знаний состоит из набора правил. Программа, управляющая перебором правил, называется машиной вывода. Чаще всего вывод бывает прямой (от данных к поиску цели) или обратный (от цели для ее подтверждения к данным). Данные — это исходные факты, на основании которых запускается машина вывода — программа, перебирающая правила из базы.

Продукционная модель чаще всего применяется в промышленных ЭС. Она привлекает разработчиков своей наглядностью, высокой модульностью, легкостью внесения дополнений и изменений и простотой механизма логического вывода. Имеется большое число программных средств, реализующих продукционный подход (например, язык OPS_5 [25]; «оболочки» или «пустые» ЭС — EXSYS [26], ЭКСПЕРТ [27]; инструментальные системы ПИЭС [28], СПИЭС [29]), а также промышленных ЭС на его основе (ФИАКР [30]).

4.3. Модули, управляемые образцами

В традиционном программировании команды устанавливаются в жесткой фиксированной последовательности. По умолчанию после выполнения *i*-й команды выполняется

(i+1)-я команда, если i-я команда не является командой ветвления. Все места в традиционном программировании указываются явно. Подобный способ программирования удобен в тех случаях, когда последовательность обработки мало зависит от обрабатываемых данных, т. е. тогда, когда ветвление является исключением, а не нормой. В противном случае программу лучше рассматривать как совокупность независимых модулей, управляемых образцами. На каждом шаге работы такая программа анализирует текущую ситуацию и определяет по анализу образцов, какой модуль подходит для обработки этой ситуации [7; 8].

Каждый *управляемый образцом модуль* (*УОМ*) состоит из механизмов исследования и модификации одной или нескольких структур данных. Диапазон УОМ может колебаться в широких пределах: от простого продукционного правила до процедуры произвольной степени сложности, вызываемой по образцу. Каждый УОМ на очередном шаге работы анализирует данные рабочей памяти, проверяя наличие структур, которые сопоставляются с его образцом. Системы, построенные на основе УОМ, называют *системами вывода, управляемыми образцами*. Функции управления в этих системах осуществляет интерпретатор [7; 8].

С точки зрения представления знаний подход, использующий управляемые образцами модули, можно охарактеризовать следующими особенностями:

- разделение постоянных знаний, хранимых в базе знаний, и временных знаний, хранимых в рабочей памяти;
- структурная независимость модулей, облегчающая модификацию и совершенствование системы, что чрезвычайно важно для экспертных систем, постоянно модифицирующих свои знания. Кроме того, независимость модулей упрощает объединение программ, написанных разными авторами;

- отделение схемы управления от модулей, несущих знания о проблемной области, что позволяет применять различные схемы управления.

Системы, управляемые образцами, классифицируются в соответствии с ограничениями, накладываемыми на модули. На рис. 4.1 приведена классификация систем, управляемых образцами, которая может рассматриваться и как классификация модулей, управляемых образцами [8].

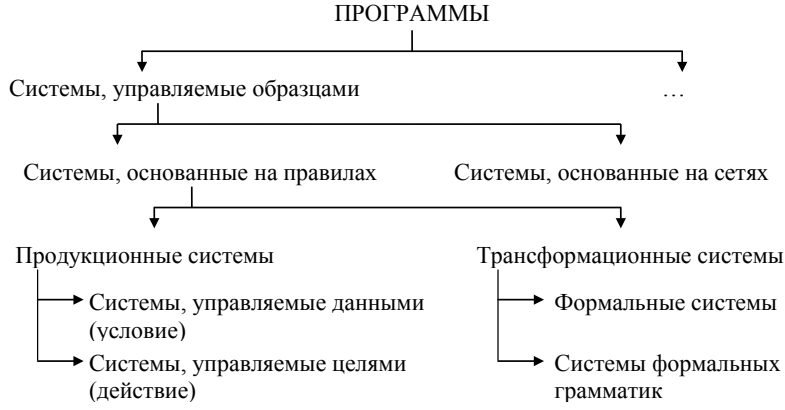


Рис. 4.1. Классификация систем, управляемых образцами

Если системы, управляемые образцами, состоят из модулей, локализованных на вершинах сети, которые активизируются сигналами, приходящими по дугам, то такие системы называются *системами, основанными на сетях* [7; 8].

Большинство систем, управляемых образцами, удовлетворяет следующему ограничению: все исследования данных рабочей памяти в каждом модуле объединены и предшествуют всем действиям по модификации данных. Таким образом, модуль разделяется на две части: *предусловие*, исследующее данные, и *действие*, модифицирующее данные. Модули,

имеющие такое деление, называют *правилами*, а системы, использующие такие правила, называют *системами, основанными на правилах.* Разделение операции исследования и модификации данных облегчает отладку системы [7; 8].

Системы, основанные на правилах, разделяются по видам правил на *продукционные* системы и *трансформационные* системы. Продукционные системы образованы из правил, в которых сопоставление и планирование (управление) являются явными функциями системы, зафиксированными в интерпретаторе. Трансформационные системы, в отличие от продукционных, могут не иметь явных функций по сопоставлению правил и управлению правилами. Примерами трансформационных систем являются формальные системы и системы формальных грамматик. Продукционные системы могут быть разделены на продукционные системы, управляемые данными (предусловиями правил), и на продукционные системы, управляемые целями (действиями правил). Традиционно под продукционными системами понимают только системы, использующие вывод, направляемый данными. Обычно предусловие (антецедент) задается в виде логической комбинации утверждений о данных рабочей памяти, а действием (консеквентом) является некоторая операция по модификации рабочей памяти. Сложность действий колеблется в значительных пределах: от простой операции присваивания до функции произвольной степени сложности [7; 8].

В продукционных системах, управляемых целями, предусловия и действия являются утверждениями о данных. Здесь вывод осуществляется в обратном направлении от утверждений, которые должны быть доказаны. Необходимо подчеркнуть, что образцы могут быть заданы как декларативно, так и процедурно [7; 8].

Итак, представление знаний в виде УОМ и продукционных правил обладает следующими достоинствами [7; 8]:

- модульностью организации знаний;
- независимостью правил, выражающих самостоятельные фрагменты знаний;
- легкостью и естественностью модификации знаний;
- отделением управляющих знаний, что позволяет применять различные управляющие стратегии;
- возможностью создания для ряда приложений управляющих механизмов для автоматического решения задач.

Основным недостатком данного подхода является его более низкая эффективность по сравнению с методами традиционного программирования. Различные авторы по-разному классифицируют продукционные системы [7; 8]. Одни относят их к декларативному представлению, другие — к процедурному или декларативно-процедурному. Вероятно, расхождения объясняются тем, насколько широко трактуется понятие «продукционное правило». По мнению авторов [7; 8], даже в самом простом продукционном правиле (т. е. правиле, не содержащем присоединенных процедур) есть элемент процедурности, так как предполагается, что правило будет использовано для выполнения некоторого действия. Именно это и отличает процедурное представление от декларативного, поскольку декларативные знания не несут никакой информации о том, как они будут использованы. В более сложных продукционных правилах степень «процедурности» еще выше. Однако в продукционных правилах и даже в модулях, управляемых образцами, есть и элемент декларативности, так как способ использования правил и модулей в самих правилах и модулях не указывается. В общем, можно считать, что, так же как и представления в виде фреймов и иерархических сетей, продукционные правила объединяют в себе свойства и декларативного, и процедурного представлений [7; 8].

4.4. Семантические модели

В основе этих моделей лежит понятие сети, образованной помеченными вершинами и дугами. Вершины сети представляют некоторые сущности (объекты, события, процессы, явления), а дуги — отношения между сущностями, которые они связывают. Пример семантической сети приведен на рис. 4.2 [24].

В качестве вершин тут выступают понятия «человек», «Иванов», «Волга», «автомобиль», «вид транспорта» и «двигатель»; в качестве дуг — отношения «это» (IS-A; АКО: A-Kind-of — является), «имеет частью» (has part), «принадлежит», «любит» [24].

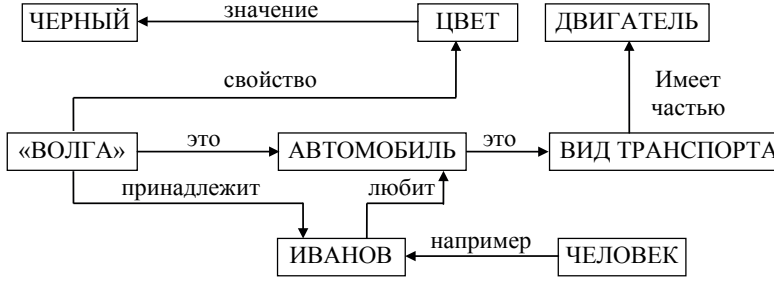


Рис. 4.2. Семантическая сеть

Характерной особенностью семантических сетей является наличие трех важнейших типов отношений: класс — элемент класса («это»); свойство — значение («цвет — красный»); пример элемента класса («например») [24].

Кроме перечисленных, в семантических сетях используются следующие типы отношений: связи типа «часть — целое» («класс — подкласс», «элемент — множество» и т. п.); функциональные связи («объект — свойство», «свойство — значение»); атрибутивные связи (количественные, временные,

пространственные); логические связи (и, или, не); лингвистические связи.

Семантические сети можно классифицировать на [24]:

- однородные (с единственным типом отношений); неоднородные (с различными типами отношений);
- бинарные (в которых отношения связывают два объекта); *n*-арные (в которых есть специальные отношения, связывающие более двух понятий).

Проблема поиска решения в базе знаний типа семантической сети сводится к поиску фрагмента сети, соответствующего некоторой подсети, представляющей вопрос.

Наложив ограничения на описания вершин и дуг, можно получить сети различного вида. Если вершины не имеют собственной внутренней структуры, то соответствующие сети называют *простыми сетями*. Если вершины обладают некоторой структурой, то такие сети называют *иерархическими сетями*. В настоящее время в большинстве приложений, использующих семантические сети, они являются иерархическими. На рис. 4.3 изображен фрагмент простой семантической сети, выражающий примерно следующую информацию: «Михаилу в течение интервала времени [t_1, t_2] принадлежат ЖИГУЛИ № 25–15». Михаил есть агент (собственник) в событии В1. Михаил является элементом (э) множества ЧЕЛОВЕК. ЖИГУЛИ № 25–15 являются элементом множества МАШИНЫ. Событие В1 является элементом множества всех событий ВЛАДЕТЬ, которое является подмножеством (п) множества СИТУАЦИИ и т. д. Использованные на рис. 4.3 дуги «э» (элемент) и «п» (подмножество) служат для выражения таксономии понятий, представленных вершинами. Важность таксономии заключается в том, что множества обычно имеют свойства, присущие всем элементам данного множества. Эти свойства связываются в сети не с конкретными элементами, а с вершинами, сопоставляемыми всему множеству. Так, дуга

«п» указывает отношение быть «подмножеством». В связи с тем, что большинство подмножеств являются различными, т. е. непересекающимися, для их представления удобно ввести дугу специального вида — «пр» (подмножество различное). Дуга «пр», идущая из вершины *x* (например, люди) в вершину *z* (например, официальные лица), указывает, что множество, представленное вершиной *x* (люди), есть подмножество множества, представленного вершиной *z* (официальные лица), и что *x* (люди) не пересекается ни с каким другим подмножеством, представляемым вершиной *r* (например, предприятия) и имеющим дугу «пр» из *r* (предприятия) в *z* (официальные лица). По аналогии можно ввести и отношение «эр» [7; 8].

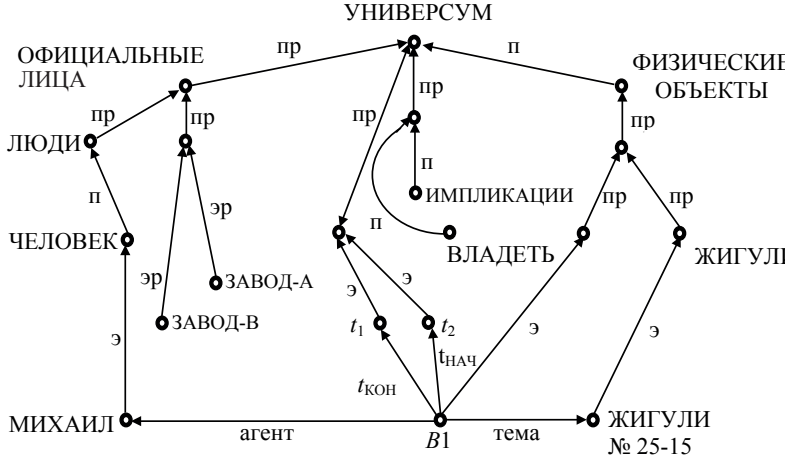


Рис. 4.3. Фрагмент семантической сети

Одно из основных отличий иерархических семантических сетей от простых семантических сетей состоит в возможности разделить сеть на подсети (пространства) и устанавливать отношения не только между вершинами, но и между пространст-

вами [9]. Все вершины и дуги являются элементами по крайней мере одного пространства. Отметим, что понятие пространства аналогично понятию скобок в математической нотации. Различные пространства, существующие в сети, могут быть упорядочены в виде *дерева пространств*, вершинам которого соответствуют пространства, а дугам — отношения «видимости».

На рис. 4.4 приведен пример дерева пространств, в соответствии с которым, например, из пространства P_6 (пространство-потомок), видимы все вершины и дуги, лежащие в пространствах-предках P_4, P_2, P_0, а остальные пространства невидимы. Отношение «видимости» позволяет сгруппировать пространства в упорядоченные множества — «перспективы». Перспектива обычно используется для ограничения сетевых сущностей, «видимых» некоторой процедурой, работающей с сетью. Обычно в перспективу включают не любые, а иерархически сгруппированные пространства. При графическом изображении иерархических сетей обычно используют следующие соглашения [7; 8]:

1) вершины и дуги, лежащие в одном пространстве, ограничиваются на рисунках многоугольником (обычно прямоугольником);
2) дуга принадлежит тому пространству, в котором находится имя дуги;
3) пространство P_i (точнее, ограничивающий его многоугольник), изображаемое внутри пространства P_j, считается потомком (внутренним уровнем), т. е. из P_i «видимо» P_j. Пространство P_i может рассматриваться как супервершина, которая лежит в P_j. Свойство «невидимости» позволяет повысить эффективность операции поиска в сети. Например, при поиске конкретных фактов информация из кванторных утверждений и правил невидима, так как она заключена в пространствах, ограничивающих эти утверждения и правила [7; 8].

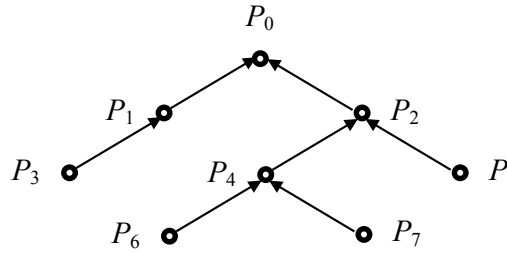


Рис. 4.4. Пример разбиения сети на пространства

При необходимости в иерархических сетях можно представить любые логические связки и кванторы. На рис. 4.5 и 4.6 приведено представление выражений, содержащих соответственно импликацию и квантор всеобщности (выраженный через импликацию). Кроме представления логических связок и кванторов, сеть может быть использована также для кодирования других структур высших порядков.

При решении многих конкретных задач представление знаний только в виде семантических сетей оказывается неудобным или неэффективным. По этой причине в семантических сетях вводят механизм процедурных присоединений.

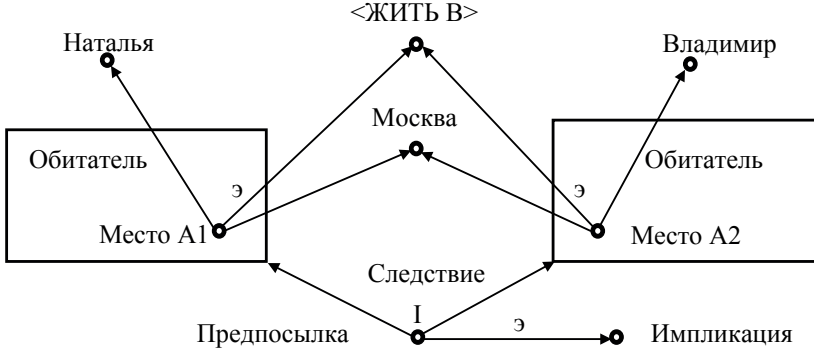
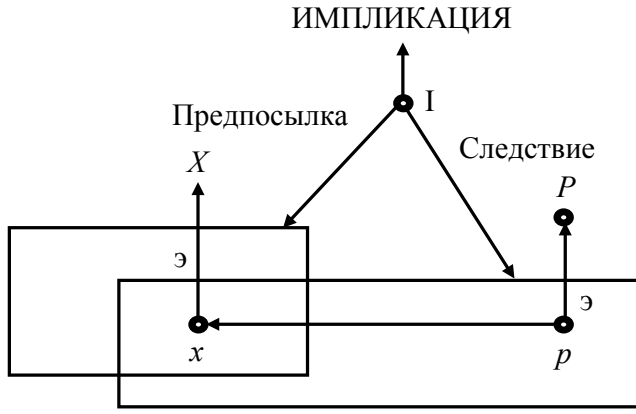


Рис. 4.5. Сетевое представление импликации



$$(\forall\, x \in X)\, P(x) \;\equiv\; \forall\, x\, ((x \in X) \rightarrow P(x))$$

Рис. 4.6. Изображение в сети квантора всеобщности с использованием импликации

Основным преимуществом семантической модели является то, что она больше других соответствует современным представлениям об организации долговременной памяти человека и была предложена психологом Куиллианом. Термин «семантическая» означает «смысловая», а сама семантика — это наука, устанавливающая отношения между символами и объектами, которые они обозначают, т. е. наука, определяющая смысл знаков. Недостатком этой модели является сложность организации процедуры поиска вывода на семантической сети [24].

Для реализации семантических сетей существуют специальные сетевые языки, например, NET [31], ВОЛНА [32; 33] и др. Широко известны ЭС, использующие семантические сети в качестве языка представления знаний, например, PROSPECTOR, CASNET, TORUS [34; 35].

4.5. Фреймы

Стремление разработать представление, соединяющее в себе достоинства различных моделей, привело к возникновению так называемого *фрейм-представления* [36] (фрейм — от англ. *frame* — «рама» или «рамка»). По Минскому, *фрейм* — это структура данных (т. е. декларативное представление), предназначенная для представления некоторой стандартной ситуации. С каждым фреймом ассоциируется разнообразная информация (в том числе и процедуры), например, информация о том, как пользоваться данным фреймом, каковы ожидаемые результаты выполнения фрейма, что делать, если ожидания не оправдались, и т. п. Фрейм можно представить в виде сети, состоящей из вершин и отношений (дуг). «Верхние уровни» фрейма фиксированы и представляют сущности, всегда истинные в ситуации, описываемой данным фреймом. «Нижние уровни» заканчиваются *слотами* (от англ. *slot* — поле), которые заполняются конкретной информацией при вызове фрейма. Можно провести аналогию между фреймами и описанием процедур в языках программирования [7; 8].

Фрейм соответствует описанию процедуры, а означенный фрейм (фрейм-пример) соответствует вызову процедуры. Отличие фреймов от описаний процедур состоит в том, что фреймы могут вызываться не по имени, а по соответствию текущей ситуации той ситуации, которую описывает данный фрейм. Кроме того, фрейм, слоты и механизм их означивания описывают ситуацию в семантических (а не синтаксических) терминах и в метатерминах. С каждым слотом фрейма связаны описания условий, которые должны быть соблюдены, чтобы могло произойти означивание слота. В простейших случаях эти условия могут сводиться к указанию семантических категорий, которым должно удовлетворять значение слота. В более сложных случаях условия могут касаться отно-

шений между значениями, выбираемыми для нескольких слотов [7, 8].

Итак, фрейм-пример может быть представлен в виде следующей конструкции:

$$f = [< r_1, v_1 >, < r_2, v_2 >, \ldots, < r_n, v_n >],$$

где f — имя фрейма; r_i — имя слота; v_i — значение слота. В качестве значений слотов могут выступать имена других фреймов, что обеспечивает связь между фреймами (так образуются сети фреймов) [7, 8].

Ту же запись можно представить в виде таблицы [24].

Имя фрейма			
Имя слота	Тип слота	Значение слота	Присоединенная процедура

В таблице дополнительный столбец предназначен для возможного присоединения к тому или иному слоту специальных процедур, что допускается в теории фреймов, и для указания типа слота [24].

Родственные фреймы связываются в систему фреймов. Система содержит описание зависимостей (причинных, временных и т. п.) между входящими в нее фреймами. Для выражения указанных зависимостей фреймы, входящие в некоторую систему, имеют общее множество слотов. Представление зависимостей в явном виде позволяет предсказать переход от одного состояния A (выражаемого фреймом A') к другому зависимому от него состоянию B (выражаемому фреймом B') и осуществить этот переход эффективно, т. е. не вычисляя заново значений всех параметров, характеризующих состояние B, а перечислив только изменившиеся (или новые) параметры [7].

Различают *фреймы-образцы*, хранящиеся в базе знаний, и *фреймы-экземпляры*, которые создаются для отображения

реальных фактических ситуаций на основе поступающих данных [24].

Модель фрейма является достаточно универсальной, поскольку существуют не только фреймы для обозначения объектов и понятий, но и *фреймы-роли* (отец, мать, начальник, пешеход), *фреймы-сценарии* (лекция, собрание), *фреймы-ситуации* (тревога, авария, рабочий режим устройства) и др. [24].

Феноменологическая сила фрейм-представления во многом основывается на включении в него предположений и ожиданий. Слотам фрейма могут быть заранее приписаны, по умолчанию, некоторые стандартные значения. Это позволяет анализировать с помощью фреймов ситуации, в которых отсутствует упоминание о ряде деталей. Стандартные значения, присвоенные по умолчанию, не жестко связаны со своими слотами и могут быть замещены более подходящими значениями, если они найдены в обрабатываемой ситуации. Использование концепции «умолчания» часто позволяет исключить необходимость в кванторных утверждениях. Системы фреймов, в свою очередь, обычно организуют в информационно-поисковую сеть. Эта сеть используется в случаях, когда предложенный фрейм не удастся привести в соответствие с данной ситуацией, т. е. когда слотам не могут быть присвоены значения, удовлетворяющие условиям, связанным с этими слотами. В подобных ситуациях сеть используется для того, чтобы предложить какой-либо другой фрейм [7; 8].

Необходимо отметить, что фрейм-представление (так же, как декларативное и процедурное представление) является не конкретным языком для представления знаний, а некой концепцией, реализуемой по-разному в таких конкретных языках, как KRL (Knowledge Representation Language) [34], FRL (Frame Representation Language) [37], K-NET и т. п. В частности, фреймы могут быть представлены с помощью иерар-

хических семантических сетей, как это сделано в K-NET, где некоторое пространство в сети рассматривается как фрейм, а дуги, связывающие это пространство с остальной сетью, рассматриваются как слоты [7; 8].

Важнейшим свойством теории фреймов является заимствованное из теории семантических сетей наследование свойств. И во фреймах, и в семантических сетях наследование происходит по АКО-связям. Наследование свойств может быть частичным [24].

Основным преимуществом фреймов как модели представления знаний является то, что она отражает концептуальную основу организации памяти человека, а также ее гибкость и наглядность [38].

Специальные языки представления знаний в сетях фреймов (FRL, KRL и др.) позволяют эффективно строить промышленные ЭС. Широко известны такие фрейм-ориентированные ЭС, как ANALIST, МОДИС [39].

4.6. Объектно-ориентированный подход

Наиболее развитым способом представления знаний в ЭС является объектно-ориентированная парадигма. Этот подход является развитием фреймового представления. В его основе лежат понятия *объект* и *класс* [7, гл. 5]; [7, прил. 1]. В реальном мире, а точнее в интересующей разработчика предметной области, в качестве объектов могут рассматриваться конкретные предметы, а также абстрактные или реальные сущности. Например, объектами могут быть покупатель, фирма, производящая определенные товары, банк, заказ на поставку. Объект обладает индивидуальностью и поведением, имеет

атрибуты, значения которых определяют его состояние. Так, конкретный покупатель, делая заказ, может оказаться в состоянии, когда денег на его счете не хватает для оплаты, а его поведение в этом случае заключается в обращении в банк за кредитом [7].

Каждый объект является представителем некоторого класса однотипных объектов. Класс определяет общие свойства для всех его объектов. К таким свойствам относятся состав и структура данных, описывающих атрибуты класса и соответствующих объектов, и совокупность методов — процедур, определяющих взаимодействие объектов этого класса с внешней средой. Например, описание класса «магазины» может включать такие атрибуты, определяющие состояние объектов, как название и адрес, которые индивидуальны для каждого объекта этого класса — конкретного магазина, штат сотрудников, размер текущего счета, а также методы: формирование заказов на поставку товаров, передача товара со склада в торговую секцию и т. д. [7].

Объекты и классы обладают характерными свойствами, которые активно используются при объектно-ориентированном подходе и во многом определяют его преимущества. К этим свойствам относятся следующие [7]:

- *инкапсуляция* — скрытие информации [7, гл. 5]. При объектно-ориентированном программировании имеется возможность запретить любой доступ к атрибутам объектов, доступ возможен только через его методы. Внутренняя структура объекта в этом случае скрыта от пользователя, объекты можно считать самостоятельными сущностями, отделенными от внешнего мира. Для того чтобы объект произвел некоторое действие, ему извне необходимо послать сообщение, которое инициирует выполнение нужного метода. Инкапсуляция позволяет изменять реализацию любого класса объектов без

опасения, что это вызовет нежелательные побочные эффекты в программной системе. Тем самым упрощается процесс исправления ошибок и модификации программ;

- *наследование* — возможность создавать из классов новые классы по принципу «от общего к частному». Наследование позволяет новым классам при сохранении всех свойств классов-родителей (называемых в дальнейшем суперклассами) добавлять свои черты, отражающие их индивидуальность. С точки зрения программиста новый класс должен содержать только коды и данные для новых или изменяющихся методов. Сообщения, обработка которых не обеспечивается собственными методами класса, передаются суперклассу. Наследование позволяет создавать иерархии классов и является эффективным средством внесения изменений и дополнений в программные системы [7];
- *полиморфизм* — способность объектов выбирать метод на основе типов данных, принимаемых в сообщении. Каждый объект может реагировать по-своему на одно и то же сообщение. Полиморфизм позволяет упростить исходные тексты программ, обеспечивает их развитие за счет введения новых методов обработки [7].

Итак, объектно-ориентированный подход заключается в представлении системы в виде совокупности классов и объектов предметной среды. При этом иерархический характер сложной системы отражается в виде иерархии классов, а ее функционирование рассматривается как взаимодействие объектов, с которыми ассоциируются, например, продукционные правила. Ассоциирование продукционных правил ЭС с иерархией классов осуществляется за счет использования общих правил, в качестве префикса которых обычно используется ссылка на класс, к которому данное правило применимо. Указанный префикс, с точки зрения декларативного

представления знаний, семантически подобен квантору всеобщности в исчислении предикатов [7].

4.7. Практика использования моделей представления знаний в экспертных системах

Для того чтобы охарактеризовать представление знаний, используемое в некоторой системе, необходимо определить представление данных в рабочей памяти и представление знаний в базе знаний. В большинстве систем из соображений эффективности в базе знаний хранятся не только изолированные правила (как предписывает теория), но и некоторая связная модель, характеризующая проблемную среду в целом. В связи с разным назначением правил и модели обычно для их задания используются различные представления. Ниже будут рассмотрены наиболее типичные подходы, используемые в экспертных системах при представлении данных, правил и моделей проблемной области [7].

Применение продукционных правил

При этом подходе данные в рабочей памяти представляются в виде изолированных троек: «объект — атрибут — значение». С каждой тройкой связан коэффициент определенности. Иерархия объектов задается с помощью дерева контекстов. Это дерево может рассматриваться как фрейм, который обеспечивает механизм наследования. В дереве контекстов каждому объекту приписаны соответствующие ему атрибуты. Впервые этот подход был использован в ЭС MYCIN [8]. Ниже приведен способ представления продукционных правил в типичной ЭС [7]:

<правило>::= (ЕСЛИ <условие> ТО <действие>
ИНАЧЕ <действие>)

<условие>::= (И {<предложение>}*)
<предложение>::= (ИЛИ {<предложения>}*) |
(<предикат><тройка>)
<тройка>::= (<объект><атрибут><значение>)
<действие>::= {<заключение>}* | <процедура>
<заключение>::= (<тройка><коэффициент определенности>)

Здесь при истинности условия выполняется действие, стоящее за указателем ТО, а при ложности — действие, стоящее за указателем ИНАЧЕ. Сущности, помеченные звездочкой, могут появиться в правиле один или более раз. Например, условие есть конъюнкция одного или более предложений, а предложение есть либо дизъюнкция одного или более предложений, либо предикат, примененный к тройке: «объект — атрибут — значение».

Использование семантических сетей

При этом подходе (см., например, ЭС PROSPECTOR [40]) данные в рабочей памяти и базе знаний представлены не в виде изолированных троек «объект — атрибут — значение», а в виде семантической сети. Отметим, что семантическая сеть может рассматриваться как обобщение представления в виде троек, допускающее представление любого *n*-арного отношения над произвольным числом объектов. Один из возможных способов представления правил в семантической сети [7]:

<правило>::= (ЕСЛИ <условие> ТО <мера И> <мера Л>
< действие>)
<условие>::= <утверждение>
<утверждение>::= <логическое утверждение> |
<описательное утверждение>
<логическое утверждение>::= (И {<утверждение>}*) |
(ИЛИ {<утверждение >}*) |
(НЕТ <утверждение>)
<действие>::= <описательное утверждение>

Правила связывают условие и действие, являющиеся утвер-

ждениями о проблемной области. Утверждения могут быть истинными и ложными. Утверждения делятся на логические и описательные. Логическое утверждение является булевской комбинацией других утверждений. Действие и терминальная компонента логического утверждения всегда являются описательными утверждениями, которые в общем случае представляют собой фрагмент семантической сети [7].

Для учета неопределенности во многих ЭС вводится коэффициент определенности утверждений. Коэффициент определенности логического утверждения вычисляется по правилам логики размытых множеств [41]. Коэффициент определенности описательных утверждений либо указывается пользователем (для исходных данных), либо (для выведенных утверждений) вычисляется по правилу Байеса [42]. В случае неопределенности каждое правило содержит две меры, используемые для модификации коэффициента определенности утверждения, выведенного этим правилом. Первая из мер используется в том случае, если условие определено как истинное, а вторая — если условие ложно. Условие оценивается как истинное, если его коэффициент определенности лежит в диапазоне: (0;+1]. При изменении коэффициента определенности некоторого условия применяется правило, и на основе процедуры Байеса подсчитывается коэффициент определенности действия (описательного утверждения) соответствующего правила [7].

Использование фреймов

При этом подходе (см., например, ЭС RLL [43]) в виде фреймов обычно представляются все виды данных и знаний, более сложных, чем список значений: объекты, правила, слоты фреймов, механизмы выводов и т. п. Фреймы организуются в сеть. Ниже приведен пример представления правила в виде фрейма [7]:

Тип *Правило*

Описание	Сообщить пользователю об опасности для его здоровья, если имеется химическая токсичность
ЕСЛИ потенциально релевантен	(Химическая активность высока?)
ЕСЛИ истинно релевантен	(Близость химически активного места и пользователя?)
ТО сказать пользователю	«Не дышать, химически активные вещества!»
ТО добавить к агенде	(<соответствующие системные указания>)
Приоритет	Высокий
Среднее время выполнения:	0,1 с
Частота использования:	Рассматривалось 985 раз, использовалось 4 раза

Представление правила в виде фрейма имеет следующие отличия [7]:

1) наличие не одного, а нескольких условий и действий для разных уровней анализа правила;
2) наличие большого количества описательных (невыполняемых) слотов, которые могут быть использованы метаправилами для управления. Необходимо обратить внимание на то, что информация о правиле хранится в нескольких слотах. Это позволяет использовать различные интерпретаторы правил в зависимости от целей разработчика. Например, один дешевый интерпретатор, предназначенный для быстрой обработки, может на основе анализа слота «ЕСЛИ потенциально

релевантен» определять число подходящих правил и уже затем принимать решение в зависимости от полученного результата. Второй, более традиционный интерпретатор оценивает все слоты «ЕСЛИ» всех правил и затем, если число подходящих правил больше одного, будет разрешать конфликт, выбирая единственное правило. Третий интерпретатор, оценив сначала, сколько времени имеется на решение задачи, может устранить из рассмотрения все правила, имеющие высокое значение слота «Среднее время выполнения», проанализировать далее слоты «Если потенциально релевантен» и, если правил все еще много, устранить те из них, которые определяют новые фреймы, а для оставшихся правил оценить все слоты «ЕСЛИ» и выбрать единственное правило.

Использование управляемых образцами модулей

Механизм использования управляемых образцами модулей проиллюстрируем на примере инструментальной системы HEARSAY-III [44], предназначенной для проектирования ЭС и являющейся модификацией системы понимания речи HEARSAY-II [7].

Все рассмотренные ранее способы представления знаний использовали частный случай управляемых образцами модулей. Действительно, каждый модуль представлялся в виде продукционного правила. Сложность правил была весьма ограничена, что позволяло выразить их в виде, понятном эксперту. Если же преобразования, выполняемые модулем, очень сложны, то для их представления приходится прибегать к процедурной форме. Стремление сохранить независимость модулей друг от друга привело к созданию в HEARSAY-III схемы, обеспечивающей взаимодействие модулей не непосредственно, а только через рабочую память, называемую «классная доска».

Модули в HEARSAY-III называются источниками знаний (ИЗ). Каждый источник знания состоит из программы-условия, которая определяет, применим ли ИЗ к текущему состоянию классной доски, и программы-действия, производящей результаты [7].

Классная доска разделена на несколько уровней, на каждом из которых обрабатываются данные определенного вида. Так, в системе HEARSAY-II выделены следующие уровни: предложение, словосочетание, слово, слог, фонема и т. д. Поиск решения рассматривается системой как итеративный процесс, состоящий из выдвижения гипотез и проверки их правдоподобности. Текущее состояние решения представляется в виде гипотез на классной доске. Гипотеза представляет собой интерпретацию некоторой части устного высказывания на определенном уровне. Гипотезы различных уровней объединены в направленный граф (сеть), что позволяет описывать гипотезы высшего уровня через гипотезы более низкого уровня [7].

Итак, в HEARSAY-III рабочая память представляется в виде сети, а знания о проблемной среде — в виде модулей, вызываемых по образцу. Использование программ, вызываемых по образцу, является шагом в направлении к процедурному представлению с попыткой сохранить независимость источников знания. Подобный подход (в отличие от использования продукций и сетей) позволяет решать значительно более сложные задачи, но уменьшает возможности по объяснению и приобретению новых знаний. Использование программ, вызываемых по образцу, требует разработки для каждой предметной области своего специфического решателя, осуществляющего планирование процесса решения и использование знаний. Применение правил в виде продукций, фреймов, сетей позволяет создавать системы, ориентированные на определенный класс задач, сохранив способности к объяснению и приобретению знаний. Однако малая мощность подобных

правил приводит к резкому снижению эффективности при решении сложных задач. Так, например, экспериментальная попытка представить часть HEARSAY-II в виде продукционных правил привела к замедлению работы примерно в 1000 раз. Общим для всех рассмотренных подходов является использование образцов при вызове модуля или правила [7].

Смешанные представления (объекты и правила)

Как правило, в экспертных системах используется не одно, а несколько представлений. Исполняемые утверждения представляются либо в виде продукционных правил, либо в виде модулей (процедур), вызываемых по образцу. Для представления модели предметной области используются объектный подход или сетевые модели (семантические сети и фреймы) [7].

Главное преимущество использования объектно-ориентированного программирования при разработке систем обработки данных заключается в поддержке методов, облегчающих повторное использование кода. Однако, как отмечают многие исследователи, эффект от внедрения объектно-ориентированной технологии программирования начинает проявляться лишь через 5–8 лет. Это обусловлено необходимостью накопления опыта разработок и формирования устойчивой и достаточно гибкой иерархии классов. Очевидно, что подобные издержки неприемлемы для инструментальных средств инженерии знаний, где одним из определяющих требований является необходимость создания «быстрого прототипа». Поэтому объектно-ориентированный инструментарий для создания систем, основанных на знаниях, должен включать и библиотеку стандартных, но достаточно легко модифицируемых объектов [7].

Применение объектно-ориентированного подхода в системах инженерии знаний выводит на первый план другую его особенность, а именно возможность естественной деком-

позиции задачи на совокупность подзадач, представляемых достаточно автономными агентами, работающими со знаниями. На сегодняшний день это единственная практическая возможность работы в условиях экспоненциального роста сложности (количества взаимосвязей), характерного для систем, использующих знания. Так, практически все инструментальные средства для создания динамических ЭС поддерживают объектно-ориентированный подход к проектированию систем, объединенный с правилами [7, гл. 9].

5. ОСОБЕННОСТИ ПРЕДСТАВЛЕНИЯ ЗНАНИЙ В СУЩЕСТВУЮЩИХ ЭКСПЕРТНЫХ СИСТЕМАХ И ИНСТРУМЕНТАЛЬНЫХ СРЕДСТВАХ ДЛЯ ИХ РАЗРАБОТКИ

Рассмотрим особенности представления знаний (состав знаний и способ их представления) в существующих известных экспертных системах и инструментальных средствах для их разработки [45–48].

5.1. Промышленная экспертная система DENDRAL

Система [49; 50] разработана в рамках Стэндфордского проекта, продолжавшегося 16 лет (с 1965 г.). Областью приложения DENDRAL является химия. Задача системы состоит в том, чтобы на основе химической формулы и масс-спектрограммы вывести возможные структуры молекулы. Эта задача

относится к классу задач интерпретации. DENDRAL решает практические задачи, требующие высокой классификации.

Знания состоят из химических структур, правил и ограничений. Химические структуры представляются в виде графов, состоящих из вершин (атомов) и дуг (связей между атомами). Ограничения представляются в виде подграфов с указанием о том, запрещены они или возможны. Правила, на основании которых генерируются структуры молекул, имеют форму «ситуация → действие», где «ситуация» и «действие» представлены в виде подграфов.

Процедура вывода представляет собой эвристический поиск, происходящий в три этапа: «план — генерация — проверка». На этапе «план» определяются пространство поиска и ограничения, соответствующие входным данным. На этапе «генерация» порождаются все возможные структуры, удовлетворяющие ограничениям. При поиске используется стратегия, «направляемая данными». Пользователь может задать дополнительные ограничения. На этапе «проверка» осуществляется отбор лучших из предложенных решений, т. е. выбираются те из полученных структур, которые наиболее точно соответствуют исходным данным.

В первых редакциях система DENDRAL имела минимальные объяснительные способности и простейший язык общения. Процесс приобретения знаний весьма трудоемок и слабо автоматизирован. Для устранения этого недостатка была разработана система MetaDENDRAL, автоматизирующая процесс приобретения правил DENDRAL.

5.2. Системы MYCIN, TEIRESIAS, EMYCIN

Система MYCIN [51] находится на стадии действующего прототипа. Областью приложения системы MYCIN является медицина. Система решает задачи постановки диагноза

и определения методов лечения инфекционных заболеваний крови. По качеству решений система не уступает эксперту-человеку.

Знания в MYCIN разделяются на факты и продукционные правила (продукции). Общее количество правил около 400. Факты представляются в виде троек («объект — атрибут — значение»), снабженных коэффициентом определенности. Продукции представлены в форме «условие → действие», где «условие» есть булевское выражение предикатных функций, применимых к фактам, а «действие» — факт или операция над фактом. Факту, полученному в результате выполнения продукции, присваивается коэффициент определенности, который вырабатывается по определенным правилам в зависимости от значений коэффициентов определенности у фактов, входящих в условие. Если этот коэффициент меньше некоторого положительного порогового значения, то его значение приравнивается нулю.

Процедура вывода реализуется в виде исчерпывающего поиска, «направляемого целями». При постановке диагноза для каждого из 100 известных ей заболеваний система просматривает правые части продукций и выделяет те из них, которые содержат заключения о текущей цели (заболевании). Для таких продукций система пытается вычислить истинность условий. Если значение некоторого условия неизвестно, то входящие в него факты рассматриваются как подцели и процесс повторяется. В связи с тем, что осуществляется исчерпывающий поиск, в результате будет найден (если, конечно, он существует) путь, приводящий от фактов, характеризующих условие задачи, к диагнозу.

Диалог с пользователем-специалистом ведется на ограниченном английском языке. От пользователя не требуется знаний в области программирования. В связи с ограниченностью предметной области используется простой лингвистиче-

ский процессор, поскольку оказалось достаточным понимать смысл предложений на уровне ключевых слов. Система обладает объяснительными способностями: анализируя процесс получения решения, она может отвечать на вопросы о том, ПОЧЕМУ был использован некий факт или КАК этот факт был установлен. Система обладает способностью модифицировать старые правила и приобретать новые. Для развития объяснительных способностей системы MYCIN и автоматизации приобретения знаний была построена инструментальная система TEIRESIAS [52–54].

Стремление использовать подход, примененный в MYCIN, для более широкого круга приложений привело к созданию инструментальной системы EMYCIN (Empty MYCIN, т. е. пустой МИЦИН), которая является версией MYCIN, не зависящей от проблемной области. Таким образом, EMYCIN [55] является базовой системой, предназначенной, в первую очередь, для решения задач диагностики. EMYCIN позволяет применить процедуры вывода к такой предметной области, знания о которой могут быть записаны в виде правил MYCIN. Система EMYCIN находится на исследовательской стадии.

5.3. Системы PROSPECTOR и KAS

ЭС PROSPECTOR достигла стадии промышленной системы. Областью приложения системы PROSPECTOR [56; 40] является геология. Задача системы — помочь геологу определить наличие месторождения руды заданного вида на основе геологических данных. Качество решений, получаемых системой, не уступает качеству решений, даваемых экспертом-геологом. Знания об области экспертизы представлены в виде моделей, каждая из которых соответствует месторождению определенного типа. Правила имеют вид «условие → дейст-

вие (*X*><*Y*)». Условие и действие суть утверждения о проблемной области, которые могут быть истинными (от 1 до 0) и ложными (от 0 до –1). Условие является булевской комбинацией утверждений. Действие состоит из одного утверждения. Все утверждения представляются в виде семантических сетей с пространствами [57, 9], которые представляют собой обобщение троек, использованных в MYCIN.

Для работы с неопределенными знаниями с каждым утверждением связано значение вероятности, т. е. мера, оценивающая степень истинности утверждения. Вероятность логического утверждения (т. е. утверждения, содержащего булевские операции) вычисляется исходя из вероятности компонент этого утверждения по правилам Заде для размытых множеств [41]. Вероятность простого утверждения (т. е. утверждения, не содержащего булевские операции) определяется или из ответа пользователя, или по правилу Байеса [42]. <*X*> и <*Y*> в правиле указывают, как модифицировать вероятность утверждения в действии правила: <*X*> используется тогда, когда условие истинно (т. е. вероятность условия находится в диапазоне 0–1), <*Y*> — когда условие ложно (т. е. вероятность в диапазоне 0 — (–1)).

Процедура вывода в системе может осуществлять поиск решения не только с помощью стратегии, «направляемой целями» (как в MYCIN), но и с помощью стратегии, «направляемой данными». При поиске, «направляемом целями», вместо исчерпывающего поиска в ширину (как это делается в MYCIN) используется метод эвристических оценочных функций.

Общение системы с пользователем осуществляется в виде обмена простыми предложениями английского языка. Объяснительные способности подобны способностям MYCIN. Для упрощения процесса приобретения знаний системой PROSPECTOR разработана система KAS. В процессе приобретения знаний KAS подсказывает пользователю пропущенные

им обязательные компоненты новых знаний. Ядром KAS является сетевой редактор, в котором имеются знания о процессах, происходящих в PROSPECTOR. Именно эта способность KAS позволяет ей при приобретении знаний осуществлять их проверку с точки зрения как формы, так и содержания.

KAS может рассматриваться как не зависящая от проблемной области версия PROSPECTOR, она соотносится с PROSPECTOR так же, как EMYCIN с MYCIN. KAS позволяет использовать процедуры вывода PROSPECTOR в проблемных областях, знания о которых могут быть представлены в виде правил, используемых в системе PROSPECTOR.

5.4. Экспертная система CADUCEUS

Первоначальное название этой системы INTERNIST [58]. Система находится на стадии действующего прототипа. Областью экспертизы является диагностика внутренних болезней человека. Система содержит информацию о 500 болезнях, что составляет 80% всех известных терапевтических заболеваний. База данных CADUCEUS — самая большая среди баз данных существующих ЭС. В системе хранятся знания приблизительно о 100000 ассоциативных связях между симптомами и болезнями. Система используется в клинической практике и для обучения студентов-медиков. Качество решений системы высокое. Знания системы включают два основных типа данных: болезни и их проявления (анамнез, симптомы, признаки, лабораторные данные). Между этими типами данных установлен ряд отношений. Каждой болезни сопоставлен ассоциативный список ее проявлений. Существует и обратный список, т. е. для каждого проявления указаны болезни, при которых оно встречается. Связям между болезнями и проявлениями присвоены весовые коэффициенты (от 1 до 5). Бо-

лезни организованы в иерархическое дерево, а также связаны причинными, временными и ассоциативными отношениями. Знания представляются в виде сетей с использованием коэффициента определенности. При поиске решения система комбинирует стратегию, «направляемую данными», со стратегией, «направляемой целями». Данные о пациенте используются для порождения некоторых гипотез (целей). Эти гипотезы используются для порождения других гипотез, которые либо подтверждаются, либо используются для изменения исходных гипотез. В процессе поиска решения используются знания о таксономии болезней и о причинных взаимоотношениях между ними. В рассуждениях системы учитывается фактор неопределенности знаний.

Интерфейс с пользователем довольно удобен. Объяснительные способности минимальны. Система ориентирована на возможное расширение списка болезней, симптомов и взаимосвязей между ними. При добавлении новых знаний модификация связей осуществляется автоматически.

5.5. Коммерческая экспертная система R1

Областью приложения системы R1 является вычислительная техника [59]. На основании заказа пользователя, приобретающего требуемую ему конфигурацию вычислительной системы VAX-11/780 фирмы DEC, система R1 выполняет следующие функции:

1) определяет, не содержит ли заказ пользователя несовместимых компонент системы VAX, и выявляет компоненты, недостающие для функционирования заказываемой конфигурации;

2) выдает в виде диаграммы конечную конфигурацию VAX, которая используется техническими службами при установке системы заказчику;
3) учитывает при построении диаграммы накладываемые заказчиком ограничения (на желаемый порядок расположения компонент, на типы и длину кабелей, связывающих устройства, и т. п.).

Сложность задач, решаемых системой R1, обусловлена сложностью системы VAX-11, состоящей из 420 компонент, и многочисленностью правил, ограничивающих возможные способы взаимодействия компонент. Система R1 превосходит эксперта-человека по качеству работы и по времени решения задачи (R1 безошибочно определяет конфигурацию за 2,5 минуты, а эксперт-человек, затрачивая на работу несколько часов, допускает ошибки, обнаруживаемые только после установки системы).

Знания в системе представляются в виде правил: «условие → действие». В системе используется примерно 3000 правил. Правила можно рассматривать как операции, преобразующие состояния. Условная часть каждого правила описывает свойства состояния, к которому правило применимо. Часть «действие» определяет, как текущее состояние должно быть преобразовано.

Особенность процедур вывода в системе R1 состоит в том, что поиск решения осуществляется, как правило, методом сопоставления, а не методом «генерация и проверка», т. е. используется безвозвратная стратегия [60]. При безвозвратном управлении применимое правило используется необратимо. Говоря другими словами, вместо генерации нескольких гипотез и выбора среди них одной, приводящей к решению, система исследует проблемную область и генерирует единственное приемлемое решение. Метод сопоставления применим в тех случаях, когда знания, используемые на каждом шаге,

достаточны для того, чтобы отличить приемлемое решение от неприемлемого. Почти все задачи, решаемые системой R1, удовлетворяют этому требованию, что позволило избежать перебора при поиске решения.

5.6. Планировщик STRIPS

Программа STRIPS [61] демонстрирует один из подходов к представлению проблем. Наименование программы — аббревиатура от Stanford Research Institute Problem Solver (решатель проблем Станфордского исследовательского института). Программа предназначалась для решения проблемы формирования плана поведения робота, перемещающего предметы через множество (анфиладу) помещений. Программа STRIPS оказала очень большое влияние на последующие разработки в области искусственного интеллекта, и те базовые методики представления знаний, которые были в ней использованы для формирования действий, не утратили своей актуальности до настоящего времени.

Текущее состояние окружающей среды — помещений и предметов в них — представляется набором выражений «предикат-аргумент», которые в совокупности образуют модель мира. Так, набор формул

$$W = \{at\ (робот, комната\ А), at\ (ящик\ 1, комната\ Б), at\ (ящик\ 2, комната\ В)\}$$

означает, что робот находится в комнате А, и имеются два ящика, один из которых находится в комнате Б, а второй — в комнате В.

Действия, которые может выполнить робот, принимают форму операторов, приложимых к текущей модели мира. Эти операторы позволяют добавить в модель некоторые факты

(сведения) или изъять их из модели. Например, выполнение операции

«Переместить робот из комнаты А в комнату Б»

в модели мира приведет к формированию новой модели *W′*. При этом факт *at* (робот, комната А) будет изъят из модели, а добавлен факт *at* (робот, комната Б). В результате новая модель мира будет иметь вид

$$W' = \{at\ (робот,\ комната\ Б),\ at\ (ящик\ 1,\ комната\ Б),\ at\ (ящик\ 2,\ комната\ В)\}.$$

Следует обратить внимание на то, что в данной работе рассматриваются только символические преобразования в модели мира и не затрагивается вопрос о возможности реального перемещения робота из комнаты А в комнату Б. Обладающий интеллектом робот должен быть не только способен изменять свое реальное положение в окружающей среде, но и одновременно менять свое внутреннее представление этой среды, знать, где он сейчас находится.

Таблицы операторов и методика «средство — анализ завершения»

Допустимые операции, такие как перемещение робота из одной комнаты в другую или проталкивание объектов, кодируются в таблице операторов. Ниже показан элемент этой таблицы, соответствующий операции *push* (толкать):

push (*X*, *Y*, *Z*)

Предварительные условия *at* (*робот*, *Y*), *at* (*X*, *Y*)

Список удалений *at* (*робот*, *Y*), *at* (*X*, *Y*)

Список добавлений *at* (*робот*, *Z*), *at* (*X*, *Z*)

Здесь выражение *push* (*X*, *Y*, *Z*) означает, что объект *X* выталкивается (роботом) из положения *Y* в положение *Z*, причем *X*, *Y* и *Z* — переменные в области значений, охватывающей доступное множество объектов, в то время как робот,

комната А, ящик 1, комната Б, ящик 2, комната В — это имена конкретных объектов из этого множества.

С точки зрения программиста переменные *X*, *Y* и *Z* в определении оператора, заданном элементом таблицы, — это аналоги формальных параметров в определении процедуры, которая соответствует такому действию:

«Вытолкнуть какой-либо объект из какого-либо положения в любое другое положение, если имеют место заданные предварительные условия; затем удалить формулы, указанные в списке удаления, и добавить формулы, указанные в списке добавления».

С точки зрения логики элемент *push* таблицы операторов может быть прочитан в виде формулы, которая утверждает:

«При любых *X*, *Y* и *Z* объект *X* выталкивается из *Y* в *Z*, если робот и объект *X* находятся в *Z*, а затем состояние изменятся заменой *Y* на *Z*».

Целевое состояние также представляется формулой, например:

а1 (ящик 1, комната А), а ∧ ящик 2, комната Б).

Программа STRIPS включает множество процедур, которые выполняют различные функции, в частности: обработку списка целей; выбор очередной цели; поиск операторов, которые могут быть использованы для достижения текущей цели; анализ соответствия между целью и формулами в списке добавлений в модель; установку сформулированных предварительных условий в качестве подцелей.

Чтобы представить себе, как на практике использовать подобную структуру представлений, рассмотрим простую задачу: как готовиться к ланчу с потенциальным клиентом. Для этого, во-первых, нужно иметь в своем распоряжении определенную сумму наличных денег, чтобы расплатиться; во-вторых, нужно проголодаться, поскольку речь идет о приеме пищи. Сформулированные условия можно рассматривать в качестве предварительных для достижения цели «ланч».

Эта цель может быть представлена оператором, как показано на рис. 5.1. После завершения ланча деньги будут потрачены, а у вас исчезнет чувство голода. Эти простые факты нашли отражение в списках удалений и добавлений для оператора *have lanch*. Однако обладание известной суммой наличных денег нельзя рассматривать как естественное состояние клиента. Сначала нужно получить их в банкомате, что, в свою очередь, требует передвижения. Следовательно, нужно добавить в таблицу операторов еще два элемента — *at cash mashine* (передвижение к банкомату) и *have money* (получение наличности).

Have Lunch with Client (пригласить клиента на ланч)	
Предварительные условия	*have money* (иметь деньги) *have appetite* (иметь аппетит)
Список удалений	*have money* (иметь деньги) *have appetite* (иметь аппетит)
Список добавлений	*have client* (иметь клиента)

Have money (иметь деньги)	
Предварительные условия	*At cash machine* (у банкомата) *Have cash card* (иметь кредитную карточку)
Список удалений	*at cash machine* (у банкомата)
Список добавлений	*have money* (иметь деньги)

At cash machine (у банкомата)	
Предварительные условия	*have transport* (иметь транспорт)
Список удалений	*at work* (на рабочем месте)
Список добавлений	*at cash machine* (у банкомата)

Рис. 5.1. Таблица операторов для задачи «Ланч»

Этот простой пример обладает довольно интересными свойствами. Отметим, что формулы для модели мира в исходном состоянии

at (work), have (transport)

остаются в неприкосновенности до тех пор, пока мы явно не удалим их. Таким образом, у вас остается возможность передвигаться по городу на протяжении всего времени реализации плана, поскольку формально отсутствуют какие-либо признаки, что эта возможность может быть утеряна (например, автомобиль будет угнан, или вы попадете в дорожную аварию, или по дороге к банкомату кончится бензин). Точно так же может что-нибудь произойти и с банкоматом — он может «зажевать» карточку, или вы можете забыть вытащить ее, или может вдруг появиться механическая рука и ножницами разрезать ее. Но предполагается, что последовательность действий, представленная в сгенерированном машиной плане, не должна предвидеть такие исключительные ситуации, хотя в реальной обстановке это соблюдается далеко не всегда.

Такая стратегия «обратных» рассуждений, т. е. от целей к подцелям, чрезвычайно распространена в программах искусственного интеллекта и экспертных системах, как вы уже убедились на примере системы MYCIN. Но даже на таком ограниченном множестве операторов, как в нашем примере, может существовать несколько вариантов выполнения действий. В этом случае необходимо будет организовать какой-то механизм поиска наилучшей последовательности операторов, приводящих к достижению сформулированной цели.

По существу, в системе STRIPS при выборе операторов выполняется поиск в пространстве состояний. В результате формируется план, т. е. последовательность операторов, приводящая к достижению цели, причем за основу берется стратегия «обратного» прослеживания. Основное отличие STRIPS

от других аналогичных программ состоит в том, что вместо методики «генерация — проверка» для передвижения в пространстве состояний используется другой метод, известный как «средство — анализ завершения» (*means-ends analisys*).

В контексте нашей задачи применение методики «генерация — проверка» означает следующее: для каждого текущего состояния предпринимаются попытки использовать все возможные операторы, причем после каждой попытки анализируется, не привела ли она к желанной цели. Но такая методика явно бессмысленна, поскольку количество разнообразных операций, которые робот способен выполнить в некоторой произвольной ситуации, очень велико, причем многие из этих операций не имеют никакого отношения к достижению заданной цели. Уже после нескольких первых испытаний размерность пространства состояний увеличится и будет экспоненциально нарастать с каждым новым испытанием. Совершенно очевидно, что в данном случае нужна совершенно иная стратегия.

Основная идея, которая лежит в основе метода «средство — анализ завершения», состоит в том, чтобы с каждой новой операцией отличие между текущим состоянием и целевым уменьшалось, т. е. каждая очередная операция должна приближать нас к цели. Но это предполагает включение в рассмотрение некоторой меры для оценки «расстояния» в пространстве состояний. Такая мера очень походит на оценочную функцию. Если очередная цель сформулирована в виде

at (*ящик* 1, *комната* А),

а ящик находится в комнате Б, то перемещение робота из комнаты А в комнату В никак не «приблизит» текущее состояние к целевому. А вот перемещение робота из комнаты А в комнату Б уменьшит расстояние между текущим и целевым состоянием, поскольку робот теперь сможет на очередном шаге вытолкнуть ящик из комнаты Б в комнату А. В этом смысле

поведение робота «мотивируется» от целевого состояния к подцелям, которые могут привести к достижению сформулированной цели.

В действительности программа STRIPS считывает список целей наподобие такого:

at (*ящик* 1, *комната* А), *at* (*ящик* 2, *комната* Б),

а затем сопоставляет эти цели и список добавления в описании каждого оператора. Так, цель *at* (*ящик* 1, *комната* А) будет соответствовать элементу *at* (*X*, *Z*) в списке добавлений оператора *push* (*X*, *Y*, *Z*).

Пусть существует подстановка значений переменных

X/*ящик* 1, *Z*/*комната* А,

которая приводит к равенству выражений *at* (*ящик* 1, *комната* А) *nat* (*X*, *Z*).

Программа следующим образом формирует подцели, выбирая в качестве таковых предварительные условия оператора.

(1) Подстановкой {*X*/*ящик* 1, *Z*/*комната* А} обозначить предварительное условие, которое является производным от соответствия *at* (*ящик* 1, *комната* А) *nat* (*X*, *Z*), и получить таким образом

at (*робот*, *Y*), *at* (*ящик* 1, *Y*).

(2) Найти в модели мира формулу, которая представляла бы текущее положение ящика *a*1 (*ящик* 1, *комната* Б), сравнить ее с *at* (*ящик* 1, *Y*) и в результате этого сравнения сформулировать подстановку {*Y*/*комната* Б}, которую затем применить к уже частично означенному предварительному условию. В результате будет сформулирована очередная подцель:

at (*робот, комната* Б), *at* (*ящик* 1, *комната* Б).

Теперь первое предварительное условие даст желаемое (целевое) положение робота, а второе предварительное условие уже выполнено.

Так как таблица операторов, модель мира и цели представлены с помощью одного и того же синтаксиса в виде конструкций «предикат-аргумент», то, применяя описанную выше схему сопоставления, программа довольно просто находит, какие именно операции нужно выполнить для достижения поставленной цели. Все, что нужно для этого сделать, — просмотреть списки добавлений в описании операторов и найти в них элемент, соответствующий заданной цели, как это показано на рис. 5.1.

Подцели формулируются на основе анализа предварительных условий, заданных для операторов, означивая их подстановкой переменных из формулы модели мира. Как только выбран нужный оператор, его предварительные условия преобразуются и добавляются в список подцелей. Если в текущем состоянии можно применить не один оператор, то для выбора между «кандидатами» нужно применить какую-либо эвристику. Например, можно выбрать тот из операторов, который сулит наибольшее сокращение «расстояния» между текущим состоянием и целевым. Другой возможный вариант — операторы в таблице заранее упорядочены и нужно применять тот из них, который стоит в списке раньше.

Весь процесс решения проблемы по такой методике имеет ярко выраженный рекурсивный характер. Подцели могут, в свою очередь, приводить к формулировке подподцелей и т. д. На самом нижнем уровне окажутся подцели, которые реализуются операторами, либо не имеющими предварительных условий, либо имеющими такие предварительные условия, которые удовлетворяются тривиально.

Анализ метода представления и управления в STRIPS

Для того чтобы яснее представить себе достоинства метода представления, использованного в системе STRIPS, рассмотрим альтернативный метод. Предположим, что текущее со-

стояние окружающего мира представлено в виде двумерного массива с элементами разного размера (в таком массиве элементы верхнего уровня — ячейки — представляют различные помещения, а элементы второго уровня — объекты в этих помещениях). Такой вариант представления компактнее описательного, но он не позволяет выполнять операции сопоставления, описанные в предыдущем разделе. Можно, конечно, придумать какой-нибудь способ описания целей и операций на языке, ориентированном на работу с массивами, но тогда будут утеряны некоторые из главных достоинств рассмотренной методики.

В качестве операторов придется использовать процедуры манипуляции с элементами массивов, которые с большим трудом воспринимаются человеком, а значит, отлаживать и конструировать операторы в такой форме значительно труднее, чем в форме таблиц операторов.

Программу будет значительно сложнее модифицировать и совершенствовать. Предположим, что усложнится размещение помещений и связи между ними. В таком случае придется полностью пересмотреть и вручную скорректировать все процедуры работы с массивами помещений и объектов, поскольку изменятся размерность массива и связи между его элементами. А в системе STRIPS единственное, что нужно будет сделать в этом случае, изменить модель мира, что делается значительно проще, поскольку при этом меняется не программный код, а только описания.

Предположим теперь, что в множество целей нужно включить, например, и такую: «перенести любые три ящика в комнату А», т. е. цель задает не единственное состояние мира, а множество состояний, удовлетворяющих сформулированному условию. При такой постановке проблемы набор процедур, ориентированных на табличное представление, придется пересмотреть коренным образом. Представление на базе кон-

струкций «предикат-аргумент» позволяет выразить целевое состояние, введя в выражение переменные

at (*X*, *комната* А), *at* (*Y, комната* А), *at* (*Z*, *комната* А).

После этого можно использовать прежнюю методику поиска решения.

Поиск решения проблемы предполагает использование эвристик, поскольку, как правило, существует множество вариантов, среди которых приходится выбирать.

При единообразном представлении проще находить те операторы, которые можно применить в конкретной ситуации, и просмотреть, какой эффект даст их применение. Единообразное представление также значительно упрощает программную реализацию процесса поиска.

Часто удается достичь заданной цели, применяя методику понижения уровня сложности проблемы (*problem reduction*). При этом производится обратная трассировка проблемы — «отталкиваясь» от цели; далее выясняем, какие предварительные условия требуется удовлетворить для ее достижения, и формулируем на основе таких рассуждений более простые подцели. Этот процесс рекурсивно продолжается до тех пор, пока не будут сформулированы тривиальные подцели, достижимые с помощью простейших операций.

Язык представления, подобный тому, что используется в STRIPS, с точки зрения программной реализации, является интерпретируемым языком, т. е. трансляция с этого языка выполняется интерпретатором, программой, которая способна распознавать в операторах языка формулы, подобные

push (*ящик* 1, *комната* Б, *комната* А),

и выразить заложенный в формулах смысл в терминах выполняемых процедур. Так, смысл приведенной выше формулы интерпретируется как необходимость достичь предварительных условий

at (*робот*, *комната* Б), *at* (*ящик* 1, *комната* Б),

а затем реализовать действия, предписанные списками добавлений и исключений, т. е. добавить в модель мира состояние

at (*робот*, *комната* А), *at* (*ящик* 1, *комната* А)

и исключить из модели мира состояние

at (*робот*, *комната* Б), *at* (*ящик* 1, *комната* Б).

Такой подход к интерпретации получил наименование процедуральной семантики (*procedural semantics*), поскольку все, что известно программе о смысле формулы, а именно, какие действия ей нужно выполнить для того, чтобы формула получила значение «истина». Процедуральная семантика позволяет построить связь между мыслью и действием.

5.7. Инструментальная коммерческая система OPS_5

OPS_5 является языком для представления знаний, использующим правила [62]. В его основе лежит разработанный ранее язык OPS, сконструированный для исследований в области искусственного интеллекта и психологии сознания [63]. Первым опытом применения OPS в области разработки ЭС явилась система R1. Знания в OPS_5 состоят из элементов данных и правил. Элемент данных является вектором (списком значений) или объектом, с которым связывают пары «атрибут — значение». Правила имеют вид «условие → действие», где «условие» есть образец, задающий неполное описание элементов данных, а «действие» — одна или несколько операций над данными.

OPS_5 определяет только общую схему процедуры вывода, не определяя частных стратегий. Этот язык позволяет программисту использовать символы и представлять отношения

между символами, но ни символы, ни отношения не имеют для OPS предопределенных значений. Значения определяются с помощью продукционных правил, записанных программистом. Процедуры вывода заданы в виде интерпретатора, реализующего продукционную систему. Цикл работы интерпретатора состоит из стандартных этапов: сопоставление, разрешение конфликта, выполнение. Достоинством OPS_5 является использование единых средств для представления знаний и механизмов управления. Кроме того, в OPS_5 реализован мощный и эффективный метод сопоставления образцов [64], что позволяет решать сложные задачи с минимальным перебором. К числу недостатков OPS_5 относится отсутствие развитых средств для реализации интерфейса, объяснения и приобретения знаний.

5.8. Инструментальная система ROSIE

ROSIE является языком инженерии знаний. Система ROSIE находится на исследовательской стадии [65; 66]. Отличительным свойством ROSIE является использование в ней языка, синтаксис которого подобен синтаксису английского языка. Это обстоятельство упрощает процесс создания и ведения базы данных. Необходимо отметить, что ROSIE не является системой, понимающей естественный язык, так как она не имеет встроенных семантических знаний о лексике английского языка (за исключением небольшого количества функциональных слов). Опыт использования ROSIE показал, что даже такая ограниченная форма «понимания» значительно облегчает пользователю процесс преобразования неформальных знаний в формальную модель.

Факты хранятся в базе данных системы в декларативной форме. В этой же базе, но отдельно от данных, хранит-

ся множество правил. Факт представляется в виде n-местного отношения, записываемого с использованием синтаксиса и лексики английского языка. В системе выделено пять типов отношений, соответствующих пяти типам предложений английского языка. Правила представляются в одном из трех видов: «процедура», «действие», «условие → действие». Действия бывают простые, составные и итеративные. Пользователь может организовать выполнение правил, либо упорядочив их в явном виде, либо вызывая те правила, условия которых истинны. Правила могут применяться с использованием как стратегии, «направляемой данными», так и стратегии, «направляемой целями».

За счет использования английского синтаксиса взаимодействие с пользователем осуществляется в ROSIE в удобной форме. Объяснительные способности подобны способностям MYCIN. Основное ограничение ROSIE — недостаточные способности системы по модификации своих правил и процедур вывода.

5.9. Инструментальная система HEARSAY-III

HEARSAY-III — комплекс программных средств, независимых от проблемной области и предназначенных для построения ЭС [67; 68; 44]. HEARSAY-III находится на исследовательской стадии. По своей общей структуре HEARSAY-III является модификацией системы понимания речи HEARSAY-II [68]. HEARSAY-III управляет взаимодействием независимых источников знаний. Взаимодействие между модулями и правилами в HEARSAY-III осуществляется только через обратную область памяти, называемую «классной доской» (blackboard). По-

добно OPS_5 система HEARSAY-III имеет весьма неразвитые средства для взаимодействия с пользователем, для объяснения своих действий и для приобретения знаний. Основной причиной этих недостатков HEARSAY-III является отсутствие внешнего языка высокого уровня для представления знаний.

5.10. Инструментальная исследовательская система RLL

RLL представляет собой структурированный набор программных средств, предназначенных для конструирования, использования и модификации ЭС [43]. RLL может рассматриваться как ЭС, предметной областью которой является построение ЭС. База знаний RLL содержит общую информацию о программировании и конкретную информацию о разрабатываемой системе. Факты в RLL представляются в виде фреймов [36]. В виде фреймов представляются также все понятия, слоты, механизмы наследования, управляющие структуры и правила. Кроме слотов, соответствующих «условию» и «действию», в правилах имеются слоты, задающие различную метаинформацию), которая используется при работе системы (например, «приоритет», «среднее время работы», «дата создания правила» и т. п.). Система RLL содержит набор полезных конструкций (различные типы управляющих механизмов, слотов, схем наследования и методов ассоциаций). RLL организует свои конструкции в библиотеку и представляет пользователю средства для работы с этой библиотекой. Сила RLL обусловлена общностью используемых информационных структур и алгоритмов. RLL может рассуждать о своих собственных действиях и данных. За достигнутую гибкость RLL расплачивается памятью, а не временем.

Подобно OPS_5 и HEARSAY-III, RLL обладает недостаточно мощными средствами для взаимодействия с пользователем, для объяснения своих действий и приобретения знаний.

5.11. Инструментальный комплекс для создания статических экспертных систем (на примере интегрированного комплекса ЭКО)

Комплекс ЭКО разработан в Российском научно-исследовательском институте информационных технологий и систем автоматизированного проектирования (РосНИИ ИТ и АП) [7; 69]. Наиболее успешно комплекс применяется для создания ЭС, решающих задачи диагностики (технической и медицинской), эвристического оценивания (риска, надежности и т.д.), качественного прогнозирования, а также обучения.

Комплекс ЭКО используется для создания коммерческих и промышленных экспертных систем на персональных ЭВМ, а также для быстрого создания прототипов экспертных систем с целью определения применимости методов инженерии знаний в некоторой конкретной проблемной области.

На основе комплекса ЭКО было разработано более 100 прикладных экспертных систем, например: поиск одиночных неисправностей в персональном компьютере; оценка состояния гидротехнического сооружения (Чарвакская ГЭС); подготовка деловых писем при ведении переписки с зарубежными партнерами; проведение скрининговой оценки иммунологического статуса; оценка показаний микробиологического обследования пациента, страдающего неспецифическими хроническими заболеваниями легких; психодиагностика в психосоматике, а также другие системы.

Структура комплекса ЭКО

Комплекс ЭКО включает три компонента [7; 69].

Ядром комплекса является *интегрированная оболочка* экспертных систем ЭКО, которая обеспечивает быстрое создание эффективных приложений для решения задач анализа в статических проблемных средах типа 1 и 2 (см. гл. 2).

При разработке средств представления знаний оболочки преследовались две основные цели: эффективное решение достаточно широкого и практически значимого класса задач средствами персональных компьютеров; гибкие возможности по описанию пользовательского интерфейса и проведению консультации в конкретных приложениях. При представлении знаний в оболочке используются специализированные (частные) утверждения типа «атрибут — значение» и частные правила, что позволяет исключить ресурсоемкую операцию сопоставления по образцу и добиться эффективности разрабатываемых приложений. Выразительные возможности оболочки удалось существенно расширить за счет интегрированности, обеспечиваемой путем вызова внешних программ через сценарий консультации и стыковки с базами данных (ПИРС и dBase IV) и внешними программами. В оболочке ЭКО обеспечивается слабая структуризация БЗ за счет ее разделения на отдельные компоненты — для решения отдельных подзадач в проблемной среде — модели (понятию «модель» ЭКО соответствует понятие «модуль» базы знаний системы G2, см. раздел 5.12).

С точки зрения технологии разработки ЭС оболочка поддерживает подходы, основанные на поверхностных знаниях и структурировании процесса решения.

Оболочка функционирует в двух режимах: в режиме приобретения знаний и в режиме консультации (решения задач). В первом режиме разработчик ЭС средствами диалогового ре-

дактора вводит в БЗ описание конкретного приложения в терминах языка представления знаний оболочки. Это описание компилируется в сеть вывода с прямыми адресными ссылками на конкретные утверждения и правила. Во втором режиме оболочка решает конкретные задачи пользователя в диалоговом или пакетном режиме. При этом решения выводятся от целей к данным (обратное рассуждение).

Для расширения возможностей оболочки по работе с глубинными знаниями комплекс ЭКО может быть дополнен компонентом К-ЭКО (*конкретизатором знаний*), который позволяет описывать закономерность в проблемных средах в терминах общих (абстрактных) объектов и правил. К-ЭКО используется на этапе приобретения знаний вместо диалогового редактора оболочки для преобразования общих описаний в конкретные сети вывода, допускающие эффективный вывод решений средствами оболочки ЭКО. Таким образом, использование конкретизатора обеспечивает возможность работы с проблемными средами типа 2 (см. гл. 2).

Третий компонент комплекса — *система ИЛИС*, позволяющая создавать ЭС в статических проблемных средах за счет индуктивного обобщения данных (примеров) и предназначенная для использования в тех приложениях, где отсутствие правил, отражающих закономерности в проблемной среде, возмещается обширным экспериментальным материалом. Система ИЛИС обеспечивает автоматическое формирование простейших конкретных правил и автономное решение задач на их основе; при этом используется жесткая схема диалога с пользователем. Поскольку при создании реальных приложений эксперты представляют, как правило, и знания о закономерностях в проблемной среде, и экспериментальный материал (для решения частных подзадач), возникает необходимость в использовании правил, сформированных системой ИЛИС, в рамках более сложных средств представления зна-

ний. Комплекс ЭКО обеспечивает автоматический перевод таких правил в формат оболочки ЭКО. В результате удается получить полное (адекватное) представление реальной проблемной среды, кроме того, задать гибкое описание организации взаимодействия ЭС с конечным пользователем.

Средства представления знаний в оболочке ЭКО

База знаний (БЗ) представляет собой совокупность нескольких моделей, каждая из которых описывает отдельное конкретное приложение или его компонент [7; 69]. Отдельная модель включает описание проблемной среды и знания о порядке решения задач (сценарий консультации). Описание проблемной среды состоит из описаний атрибутов и правил вывода.

Структура базы знаний оболочки изображена на рис. 5.2. Атрибуты используются для описания состояния предметной области, например атрибуты «возраст», «диагноз» и т. д. Описание атрибута включает список возможных значений, а также некоторую лингвистическую информацию, необходимую для ведения диалога с конечным пользователем. Оболочка работает со статическими проблемными средами (значения атрибутов не изменяются в ходе решения задачи), в которых предметная область может быть описана с помощью априорно заданного набора атрибутов. Не допускается динамическое создание атрибутов во время решения задачи.

Средства комплекса позволяют представлять качественные (символьные) и количественные (числовые) характеристики предметной области. Высказывания типа «*A* есть *B*», где *A* — атрибут, представляющий качественную характеристику, а *B* — одно из его возможных значений, называются утверждениями о состоянии предметной области. Например, высказывание «диагноз — острый бронхит» является утверждением предметной области, если в ней определен атрибут «диагноз», одним из возможных значений которого является «острый

бронхит». Значения символьных атрибутов задаются разработчиком ЭС при создании системы, при этом система формирует соответствующее множество утверждений. Решение задачи сводится к получению значений некоторых целевых атрибутов (например, «диагноз» или «оценка риска») и (или) определению истинности некоторых целевых утверждений.

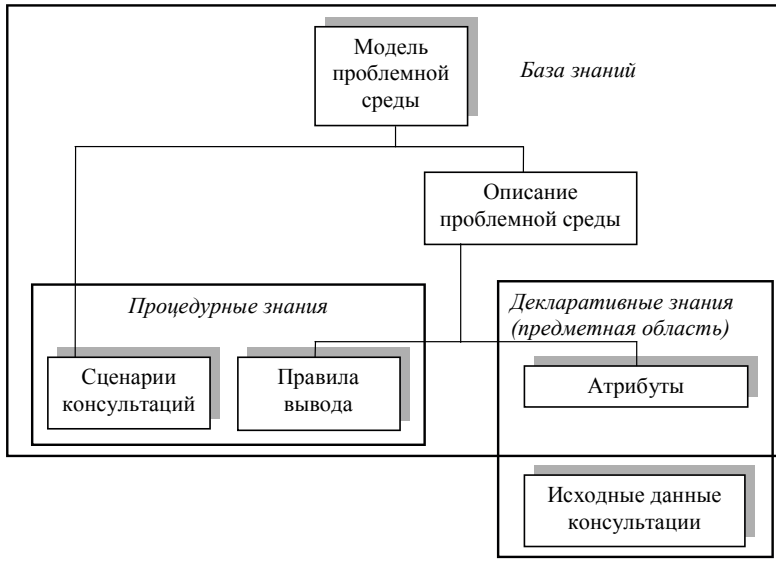


Рис. 5.2. Структура базы знаний оболочки ЭКО

Оболочка позволяет работать с неточно и нечетко определенными знаниями о качественных характеристиках предметной области. С каждым утверждением о состоянии предметной области связывается коэффициент определенности, который характеризует степень уверенности в его истинности. Значение символьного атрибута задается распределением коэффициента определенности по всем возможным его значениям. Вывод решений заключается в нахождении коэффициентов

определенности некоторых целевых утверждений, указываемых разработчиком ЭС. Для построения вывода в условиях неопределенности используются нечеткая логика и байесовский подход, кроме того, допускается применение произвольных эвристических подходов, предлагаемых экспертами. Вывод в условиях неопределенности может быть описан с помощью правил 6 типов, которые рассматриваются ниже.

Коэффициенты определенности утверждений — это действительные числа, принимающие значения от –5,00 до +5,00. Коэффициенту определенности D (H) утверждения H можно дать следующую содержательную интерпретацию:

если точно известно, что H истинно, *то* D (H) = 5,00;

если точно известно, что H ложно, *то* D (H) = –5,00;

если H может быть с одинаковой уверенностью истинно или ложно, *то* D (H) = 0,00;

если H скорее истинно, чем ложно, *то* $0{,}00 < D(H) < 5{,}00$, причем D (H) тем больше, чем больше уверенность в истинности H;

если H скорее ложно, чем истинно, *то* $-5{,}00 < D(H) < 0{,}00$, причем D (H) тем меньше, чем больше уверенность в ложности H.

Числовые значения коэффициентов определенности могут связываться со словесными описаниями с помощью лингвистических шкал, приписываемых символьным атрибутам или утверждениям. Например, шкала

«да = 5,00 возможно = 2,5, да или нет = 0,
маловероятно = –2,5, нет = –5»

связывает со словом «да» все значения коэффициента от 3,75 до 5, «возможно» — от 1,25 до 3,75 и т. д. Шкалы можно вводить и для числовых атрибутов, например для возраста:

«ребенок = 10, подросток = 14, молодой = 18, средний = 40,
пожилой = 60».

Шкалы применяются при формировании текстов вопросов и сообщений конечному пользователю.

Утверждения и числовые атрибуты модели называются целями, а символьные атрибуты, представляющие собой множество утверждений, называются сложными целями. Значения целей определяются с помощью правил вывода. Правило вывода указывает, каким образом можно получить значение цели по значениям других атрибутов и утверждений, называемых подцелями правила. Кроме того, правила могут описывать ввод исходных данных — обращение с вопросами к конечному пользователю или к внешним программам. В зависимости от типа цели (простая или сложная) правила делятся на простые и сложные.

Правила могут иметь условия применимости — нечеткие логические выражения, вычисляемые в момент обращения к правилу. Правило применяется только в том случае, если выполнено условие его применимости (вычисленный коэффициент определенности условия больше нуля). Определены следующие типы простых правил:

- простой вопрос;
- арифметическое правило;
- логическое правило;
- байесовское правило.

К сложным правилам относятся:

- альтернативный вопрос;
- дистрибутивный вопрос.

Правила-вопросы позволяют запрашивать данные о конкретной ситуации в предметной области — исходные данные консультации. Возможны два способа ввода этих данных: вопрос к пользователю или обмен информацией с внешними программами.

Простой вопрос позволяет получать либо значение числового атрибута, либо коэффициент определенности отдельного утверждения. Сложный опрос позволяет получить распределение коэффициентов определенности по всем возможным

значениям символьного атрибута. Альтернативный вопрос используется в тех случаях, когда известно, что символьный атрибут имеет точно одно значение из множества возможных значений. Дистрибутивный вопрос используется в тех случаях, когда символьный атрибут может иметь одновременно несколько значений или ни одного.

Арифметические правила предназначены для вычисления значений числовых атрибутов, а также для получения коэффициентов определенности утверждений по эвристическим формулам, предложенным экспертам.

Логические правила предназначены для вычисления коэффициентов определенности утверждений по формулам нечеткой логики, при этом значение логического выражения в условии правила присваивается коэффициенту определенности целевого утверждения правила.

Байесовские правила применяются для вычисления коэффициентов определенности тех утверждений, об истинности которых можно судить по выполнению ряда факторов (симптомов), имеющих разную значимость.

Для определения значения одной цели разработчик экспертной системы может задавать несколько правил, образующих в модели упорядоченный список правил вывода данной цели. Порядок правил в списке отражает порядок их рассмотрения во время решения задачи.

Сценарий консультации описывает порядок проведения консультации и представляет собой последовательность предложений, каждое из которых может иметь условие применимости. В рамках каждого предложения возможно выполнение одного из следующих действий: вывести значение цели; выдать сообщение пользователю; выдать сообщение внешней программе; сбросить выведенные результаты (СБРОС); перейти к выполнению другого предложения; принять информацию от внешней программы; передать информацию о ре-

зультатах решения внешней программе; создать контрольную точку консультации; загрузить контрольную точку; обратиться к подмодели, решающей некоторую частную подзадачу, передать ей параметры и получить выведенные в подмодели результаты; закончить консультацию с сообщением (СТОП).

Рассмотрим построение сети вывода на основе содержащихся в модели описаний правил и атрибутов. В процессе построения сети из числовых атрибутов, утверждений и правил строится сеть вывода, в явном виде включающая все связи между атрибутами и утверждениями, обусловленные правилами вывода. Сеть вывода образует граф с вершинами двух типов: вершины первого типа соответствуют простым целям (т. е. числовым атрибутам и утверждениям), а вершины второго типа — простым правилам. Дуги представляют связи между простыми целями и простыми правилами: если простая цель *G* выводится с помощью правила *R*, то в сети имеется дуга, которая соединяет вершины, соответствующие *G* и *R*, и направлена от *G* к *R*.

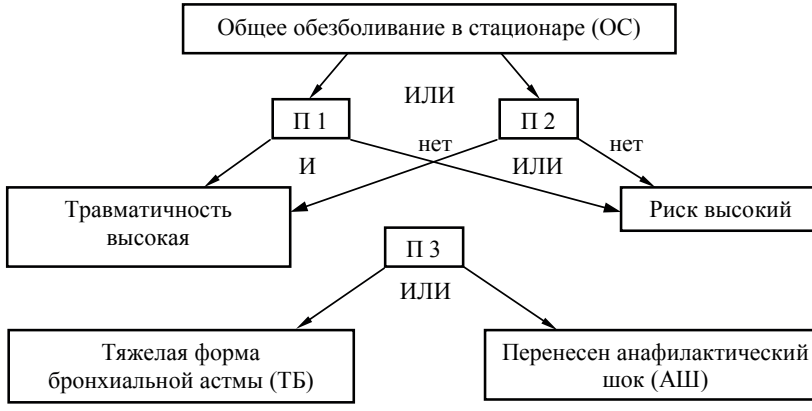
Сложные правила преобразуются в результате компиляции в совокупности простых, выводящих значения всех утверждений о целевых символьных атрибутах, соответствующих сложным правилам. Особенность таких простых правил состоит в том, что применение одного из них влечет за собой применение остальных.

Сеть вывода строится следующим образом. При вводе новой простой цели (нового числового атрибута или нового значения символьного атрибута) в сеть вывода вводится вершина, соответствующая этой цели.

При вводе простого правила, т. е. правила вывода значения простой цели, в сеть добавляются вершина, соответствующая этому правилу, и дуги, представляющие связи между простой целью и правилом, между правилом и его подцелями. При вводе сложного правила из него создается несколько простых

правил, предназначенных для вывода значений утверждений о сложной цели вводимого правила. Простые правила добавляются описанным ранее образом в сеть вывода.

Пример сети вывода и соответствующих ей правил из области анестезиологии приведен на рис. 5.3 [7, 69]. В правилах представлены рекомендации относительно целесообразности направления пациента в стационар для проведения хирургического вмешательства под общим обезболиванием. Если травматичность предполагаемого вмешательства высокая и риск, обусловленный общим состоянием пациента, высокий, то рекомендуется проведение общего обезболивания в стационаре. В противном случае (риск не является высоким или травматичность вмешательства низкая) нет показаний к проведению общего обезболивания в условиях стационара. Риск считается высоким, если пациент страдает тяжелой формой бронхиальной астмы или ранее перенес анафилактический шок.



П1: **ЕСЛИ** [Травматичность высокая] & [Риск высокий] **ТО** ОС
П2: **ЕСЛИ** ~ [Травматичность высокая] V ~ [Риск высокий] **ТО** ~ОС
П3: **ЕСЛИ** ТБ V АШ **ТО** [Риск высокий]

Рис. 5.3. Пример сети вывода

5.12. Инструментальный комплекс для создания экспертных систем реального времени (на примере интегрированной среды G2-Gensym Corp., США)

Основное предназначение программных продуктов фирмы Gensym (США) [7; 70] — помочь предприятиям сохранять и использовать знания и опыт их наиболее талантливых и квалифицированных сотрудников в интеллектуальных системах реального времени, повышающих качество продукции, надежность и безопасность производства и снижающих производственные издержки. Интегрированная среда G2 фирмы Gensym (США) и подобные ей системы предназначены для решения следующих классов задач: мониторинг в реальном масштабе времени; системы управления верхнего уровня; системы обнаружения неисправностей; диагностика; составление расписаний; планирование; оптимизация; системы — советчики оператора; системы проектирования.

Инструментальные средства фирмы Gensym явились эволюционным шагом в развитии традиционных экспертных систем от статических предметных областей к динамическим [7, 71]. Немалую долю успеха фирме Gensym обеспечили основные принципы, которых она придерживалась в своих новых разработках: проблемно-предметная ориентация; следование стандартам; независимость от вычислительной платформы; совместимость снизу-вверх с предыдущими версиями; универсальные возможности, независимые от решаемой задачи; обеспечение технологической основы для прикладных систем; комфортная среда разработки; поиск новых путей развития технологии; распределенная архитектура клиент-сервер; высокая производительность.

Основным достоинством оболочки экспертных систем G2 является возможность применять ее как интегрирующий компонент, позволяющий за счет открытости интерфейсов и поддержки широкого спектра вычислительных платформ легко объединить уже существующие, разрозненные средства автоматизации в единую комплексную систему управления, охватывающую все аспекты производственной деятельности — от формирования портфеля заказов до управления технологическим процессом и отгрузки готовой продукции. Это особенно важно для отечественных предприятий, парк технических и программных средств которых формировался по большей части бессистемно, под влиянием резких колебаний в экономике.

Успех инструментального комплекса G2 обусловлен прежде всего тем, что G2 — динамическая система в полном смысле этого слова. G2 — это объектно-ориентированная интегрированная среда для разработки и сопровождения приложений реального времени, использующих базы знаний. G2 функционирует на большинстве существующих платформ (табл. 5.1). База знаний G2 сохраняется в обычном ASCII-файле, который однозначно интерпретируется на любой из поддерживаемых платформ. Перенос приложения не требует его перекомпиляции и заключается в простом переписывании файлов. Функциональные возможности и внешний вид приложения не претерпевают при этом никаких изменений.

Таблица 5.1

Платформы, на которых функционирует G2

Фирма-производитель	Вычислительная система	Операционная среда
Digital Equipment	VAX 3xxx, 4xxx, 6xxx, 7xxx, 8xxx, 9xxx	VMS

Окончание табл. 5.1

Фирма-производитель	Вычислительная система	Операционная среда
	DECstation 3xxx, 5xxx	ULTRIX
	DEC Alpha APX	Open VMS, OSF/1, Windows NT
SUN Microsystems	SUN-4	Sun OS
	SPARC 1, 2, 10, LX, Classic	Sun OS/Solaris 1, Solaris 2.x
Hewlett Packard	HP 9000/4xx, 7xx, 8xx	HP-UX
IBM	RISC 6000	AIX
Data General	AViiON	DG/UX
Silicon Graphics	IRIS, INDIGO	IRIX
ПЭВМ	Intel 486/Pentium	Windows NT, Windows-95
Motorola	Motorola 88000	UNIX
NEC	EWS 4800	EWS-UX/V

База знаний

Все знания в G2 хранятся в двух типах файлов: базы знаний (БЗ) и библиотеки знаний (БиЗ). В файлах БЗ хранятся знания о приложениях: определения всех объектов, объекты, правила, процедуры и т. п. В файлах БиЗ хранятся общие знания, которые могут быть использованы более чем в одном приложении, например определение стандартных объектов. Файлы БЗ имеют расширение *kb* (*knowledge base*), а файлы БиЗ — *kl* (*knowledge libraries*). Файлы БЗ могут путем замены расширения преобразоваться в БиЗ и обратно.

В целях обеспечения повторной используемости приложений в G2 реализовано средство, позволяющее объединять ра-

нее созданные *kb*- и *kl*-файлы с текущим приложением. При этом G2 автоматически выявляет и выводит на дисплей конфликты в объединяемых знаниях.

Знания в G2 структурируются следующими способами: иерархия классов, иерархия модулей, иерархия рабочих пространств. Каждая из указанных иерархий может быть показана на дисплее с использованием возможности «Inspect».

Сущности и иерархия классов

Класс является основой представления знаний в G2. Понятие «класс в G2» базируется на объектно-ориентированной технологии (ООТ). Как уже указывалось ранее (см. гл. 4), использование ООТ является на текущем уровне развития ИИ и вообще программирования главной тенденцией.

В ООТ структуры данных представляются в виде классов объектов (определений объектов), имеющих определенные атрибуты. Классы наследуют атрибуты от суперклассов и передают свои атрибуты подклассам. Каждый класс (исключая корневой) может иметь конкретные экземпляры класса. В четвёртой версии G2 введен механизм множественного наследования. Теперь в системе достаточно легко произвести, например, новый класс саморегулирующихся насосов от классов контроллеров и насосов. В системе достаточно изящно решена проблема конфликтов между именами атрибутов. Использование ООТ обеспечивает следующие преимущества [7]:

1) уменьшает избыточность и упрощает определение классов, так как определяется не весь класс, а только его отличия от суперкласса;
2) позволяет использовать общие правила, процедуры, формулы, что уменьшает их количество;
3) является естественным для человека способом описания сущностей.

Класс в G2 является основой представления знаний. Все, что хранится в БЗ и чем оперирует система, является экзем-

пляром того или иного класса. Все синтаксические конструкции G2 тоже являются классами. Для сохранения общности даже базовые типы данных — символьные, числовые, булевские и истинностные значения нечеткой логики представлены соответствующими классами. Описание класса (тоже экземпляр специального класса) включает ссылку на суперкласс (is-a-иерархия) и перечень атрибутов, специфичных для класса (part-of-иерархия).

Концептуально иерархия классов G2 берет свое начало от корневого класса, именуемого «item-or-value» (сущность или значение). Класс item-or-value сам по себе не может иметь экземпляров. Однако так как он является корнем всей иерархии классов, он определяет основное поведение всех классов G2. «Item-or-value» имеет два производных класса — «value» (хотя концептуально ветвь «value» представляется классом, в действительности это типы данных G2) и item. Каждый из этих классов имеет свои производные классы. Сущность (item) является корнем разветвленной иерархии классов. Наиболее важные ветви этой иерархии могут быть сгруппированы в небольшое число категорий. Они перечислены ниже, в порядке их «видимости» для пользователя, начиная с наиболее «видимых».

Иерархия модулей и рабочих пространств

С2-приложение не представляет собой единый блок. Оно структурируется с помощью модулей и рабочих пространств на легко управляемые куски. Несмотря на то, что функции модулей и рабочих пространств похожи, между ними есть существенные различия, отмеченные ниже [7].

Приложение в G2 может быть организовано в виде одной БЗ или в виде нескольких БЗ, называемых модулями. В последнем случае говорят, что приложение модуляризировано (структурировано на модули). Модули приложения организованы в древовидную иерархию с одним модулем верхнего

уровня. Модули следующего уровня состоят из тех модулей, без которых не может работать модуль предыдущего уровня. Эти модули называют «непосредственно требуемые модули». Существуют 2 способа создать G2-приложения.

1. Разрабатывается одномодульное приложение, которое затем при необходимости разделяется на отдельные модули.
2. Приложение изначально создается как состоящее из нескольких модулей. Некоторые из этих модулей разрабатываются впервые, а другие могут выбираться из библиотеки знаний.

Структурирование приложения на модули обеспечивает следующие преимущества: позволяет разрабатывать приложение одновременно нескольким группам разработчиков; упрощает разработку, отладку и тестирование; позволяет изменять модули независимо друг от друга; упрощает повторное использование знаний.

Рабочие пространства являются контейнерным классом, в котором размещаются другие классы и их экземпляры, например объекты, связи, правила, процедуры и т. д. Каждый модуль (база знаний) может содержать любое количество рабочих пространств. Рабочие пространства образуют одну или несколько древовидных иерархий с отношением «is-a-part-of» (является частью). С каждым модулем (базой знаний) ассоциируется одно или несколько рабочих пространств верхнего (нулевого) уровня, каждое из этих рабочих пространств является корнем соответствующей древовидной иерархии. В свою очередь, с каждым объектом (определением объекта или связи), расположенным в нулевом уровне, может быть ассоциировано рабочее пространство первого уровня, связанное с ним отношением «является частью», и т. д.

Различие между модулями и рабочими пространствами состоит в следующем. Модули разделяют приложение на от-

дельные базы знаний, совместно используемые в различных приложениях. Динамические модули (аналог библиотек динамического связывания) могут подгружаться и вытесняться из оперативной памяти во время исполнения программно и одновременно использоваться несколькими приложениями. Рабочие пространства выполняют свою роль при исполнении приложения. Они содержат в себе (и в своих подпространствах) различные сущности и обеспечивают разбиение приложения на небольшие части, которые легче понять и обрабатывать. Например, весь процесс разбивается на подпроцессы, и с каждым подпроцессом ассоциируется свое подпространство.

Рабочие пространства могут устанавливаться (вручную или действием в правиле-процедуре) в активное или неактивное состояние (т. е. сущности, находящиеся в этом пространстве и в его подпространствах, становятся невидимыми для механизма вывода). Механизм активации (деактивации) рабочих пространств используется, например, при наличии альтернативных групп правил, когда активной должна быть только одна из альтернативных групп.

Кроме того, рабочие пространства используются для задания пользовательских ограничений, определяющих поведение приложения для различных категорий пользователей.

Структуры данных БЗ

Глобально сущности в БЗ G2 с точки зрения их использования могут быть разделены на структуры данных и исполняемые утверждения. Примерами первых являются объекты и их классы, связи (connection), отношения (relation), переменные, параметры, списки, массивы, рабочие пространства и т. п. Примерами вторых — правила, процедуры, формулы, функции и т. п.

Опишем наиболее важные ветви иерархии «item» [7].

Объект (object) и его подклассы. Объекты представляют объекты реального мира в приложении. Класс объектов опре-

деляет атрибуты, которые позволяют создать пиктограммы для объектов, определить их положение на схемах и отростки связей (stubs) для присоединения их к другим объектам.

Связь (connection). Класс для изображения путей между объектами. Можно создать подкласс связей для указания различных типов потоков, которые могут существовать между объектами на схеме. Например, объекты могут быть соединены водопроводными трубами и (или) проводами, передающими логические сигналы. Определив различные классы для этих связей, можно быть уверенным, что G2 будет их различать и никогда не позволит воде течь по электропроводам.

Рабочее пространство БЗ (Kb-workspace). Класс, определяющий независимый сегмент базы знаний, который может быть активирован или деактивирован. Рабочие пространства отображаются как отдельные, ограниченные рабочие области, в которых можно помещать объекты и объединять их в схемы. Можно создать связи между рабочими пространствами с помощью точек связи (connection posts). По сути, класс рабочих пространств является развитием концепции рабочей памяти в традиционных системах. Можно сказать, что рабочая память системы G2 строится на основе иерархии рабочих пространств. Иерархия рабочих пространств тесно связана с графическим представлением объектов. Рабочее пространство является контейнерным классом для экземпляров других классов. Каждый экземпляр объекта может обладать своим рабочим пространством, представляющим его внутреннюю структуру. Введение концепции рабочих пространств обеспечивает две важные функции системы G2: возможность осуществлять рассуждения на разных уровнях абстракции и возможность продолжительной (теоретически — бесконечной) работы системы без необходимости «сборки мусора» в пределах отведенного объема оперативной памяти, что очень важно для систем управления непрерывными процессами.

Классы пользовательского интерфейса (user-interface). Определяют такие элементы пользовательского интерфейса, как меню, селективные кнопки (*radio button*), сообщения (*message*), шкалы, круговые шкалы и многое другое. Можно определять подклассы класса сообщений, например, для создания сообщений со специальным способом отображения. Из всех классов пользовательского интерфейса только для сообщений есть возможность создавать производные классы.

Классы описаний классов (class definition) определяют классы, экземпляры которых содержат созданные пользователем описания классов и служат шаблонами для создания экземпляров других классов. Эти классы порождены от *класса описание (definition)*. Описание имеет три подкласса — *описание объекта (object-definition)*, *описание связи (connection-definition)* и *описание сообщения (message-definition)* в соответствии с классами, которые может определять пользователь.

Классы языка G2 (G2 language): эти классы используются для определения различных элементов языка G2, таких как правила, отношения, действия и процедуры. Нельзя создать собственные производные классы от этих классов.

С помощью G2 новые классы могут создаваться не только в процессе разработки, но и динамически, во время работы приложения. Во время исполнения приложения может быть создан, модифицирован или уничтожен экземпляр любого класса или целый класс. Это касается как объектов, так и правил и процедур. В этом смысле G2 более объектно-ориентированная система, чем даже C++. Эта возможность является частью общих возможностей G2, дополняющих описания классов и позволяющих создавать новые сущности, включая рабочие пространства, правила, связи и процедуры. G2 обеспечивает операции *create by cloning (создание клонированием)* и *change the text of (изменить текст)*, которые используются для клонирования похожего описания класса и последующе-

го редактирования его в соответствии с требованиями новых особенностей. По умолчанию динамически созданные сущности являются *временными* (*transient*), т. е. они существуют только на протяжении данного сеанса работы и не сохраняются в базе знаний. Однако описание класса должно быть *постоянной* (*permanent*) сущностью в момент создания экземпляра или производного класса. Можно использовать операцию *make permanent (сделать постоянным)* для преобразования временной сущности — описания класса в постоянную.

Все классы G2 обладают по крайней мере одним общим свойством — их экземпляры имеют графическую форму представления. Используя эти графические образы вместе с классом связей, можно строить схемы систем для любого уровня сложности. Кроме визуализации взаимодействия объектов G2 предоставляет синтаксические конструкции, позволяющие осуществлять рассуждения на основе графических схем. Например, можно проверить состояние всех вентилей, соединенных с данной емкостью, или определить температуру объекта, ближайшего к указанному.

Рассмотрим подробнее наиболее важные классы сущностей [7].

Выделяют объекты (классы), встроенные в систему и вводимые пользователем. При разработке приложения, как правило, создаются подклассы пользовательских и встроенных классов, отражающие специфику данного приложения. Среди встроенных подклассов наибольший интерес представляет подкласс объектов, включающий подклассы переменных и параметров, и подкласс связей (*connection*) и отношений (*relation*).

Объекты

Объекты в базе знаний представляют собой отображения элементов реального мира, которые будут применяться при решении поставленной перед ЭС РВ задачи. Выделяют посто-

янные и временные объекты. *Постоянные объекты* заносятся в БЗ разработчиком ЭС РВ в процессе диалога с системой, в то время как *временные объекты* создаются после выполнения специальных команд в правилах и процедурах. Временные объекты могут существовать в БЗ только в процессе работы ЭС реального времени. С каждым объектом ассоциируется таблица атрибутов, в которую заносятся существенные для решаемой задачи свойства объекта. Элемент данной таблицы представляет собой пару «атрибут — значение».

Объекты могут иметь графические образы, отображаемые на экране дисплея, называемые *пиктограммами*. На пиктограммах разработчиком могут быть выделены отдельные участки. Цвет таких участков может изменяться в результате выполнения специальных команд в правилах или процедурах. Таким способом обеспечивается высокая наглядность информации, предоставляемой лицу, работающему с ЭС реального времени.

Поскольку реальные приложения могут содержать большое количество объектов, целесообразно предоставлять возможность объединения множества объектов со схожими свойствами в классы. Классы объектов составляют иерархию, в которой определяется отношение «родительский класс — подкласс». Объекты подклассов могут наследовать атрибуты и пиктограммы родительских классов. Иерархическая упорядоченность классов значительно упрощает задачу определения новых классов в приложении. Например, атрибуты, характеризующие объекты различных классов, могут быть однократно определены в одном классе, являющемся общим родительским классом для них. Такие атрибуты будут автоматически наследоваться объектами, принадлежащими к подклассам, что снимает необходимость их повторного определения. Другим важным достоинством введения классов объектов является возможность составления правил, отно-

сящихся ко всем объектам, принадлежащим к некоторому классу (общих правил). Задача разработчика значительно упрощается за счет того, что им может быть составлен ряд общих правил, применимых к различным классам объектов приложения, а результирующая БЗ будет иметь меньший объем по сравнению с БЗ, в которой не могут применяться общие правила.

Особая роль в G2 отводится переменным. В отличие от статических систем переменные в G2 делятся на три вида: собственно переменные, параметры и простые атрибуты. *Параметры* получают значения в результате работы машины вывода или выполнения какой-либо процедуры. *Переменные* представляют измеряемые характеристики объектов реального мира и поэтому имеют специфические черты: время жизни значения и источник данных. *Время жизни значения переменной* определяет промежуток времени, в течение которого это значение актуально, по истечении этого промежутка переменная считается не имеющей значения. В отличие от переменных параметры всегда имеют значение, так как их значения либо заданы в качестве начальных значений, либо перевычислены механизмом вывода G2.

Поскольку системе может потребоваться текущее значение переменной, для каждой из них должен быть определен *источник данных (сервер данных)*. Источником данных для переменной могут служить: машина вывода, подсистема имитационного моделирования или внешний по отношению к G2 источник данных. С переменными могут быть ассоциированы формулы имитационного моделирования, в результате применения которых система также может получать значения переменных. Для параметров указанный механизм получения значений из источника данных не используется; они получают новые значения после выполнения специальных операторов в заключениях правил или процедур.

При ссылке в правиле или процедуре как для переменных, так и для параметров допустимо использование следующих выражений, отражающих динамику их значений [7]: текущее значение; значение в заданный момент времени; среднее значение за интервал времени; интеграл по интервалу времени; интерполяция значения в заданный момент времени; максимальное (минимальное) значение за интервал времени; количество собранных значений за интервал времени; скорость изменения значений в течение интервала времени; стандартное отклонение в течение интервала времени.

Очевидно, что далеко не для всех используемых в приложении значений нужно применять такой мощный инструментарий, поэтому в целях повышения эффективности функционирования системы в этих случаях используют простые атрибуты.

Связи и отношения

В G2 предусмотрены два вида взаимосвязей между объектами: связи и отношения. Под *связями* понимается взаимосвязь между двумя сущностями, задаваемая разработчиком приложения и имеющая графическое представление. В реальном физическом окружении, описываемом в G2, связи может соответствовать физическая связь между сущностями, такая как электрическое соединение или трубопровод. В G2 разработчик может задавать классы связей, ссылаться на объекты посредством указания связей, в которых они участвуют, а также делать заключения на основании наличия или отсутствия связей. Отношения, как и связи, представляют взаимосвязи между объектами. Под *отношением* понимается поименованная взаимосвязь между двумя сущностями. G2 предоставляет возможность разработчику задавать различные типы отношений. На основании наличия или отсутствия отношения между объектами могут производиться выводы. Основные отличия связей и отношений сводятся к следующему: 1) связи задаются разработчиком в про-

цессе создания ЭС, в то время как отношения устанавливаются динамически после выполнения специальных операторов в правилах или процедурах; 2) связи имеют графическое представление, в то время как отношения не отображаются на экране дисплея; 3) отношения в отличие от связей нецелесообразно сохранять в качестве постоянной части БЗ.

Исполняемые утверждения БЗ

Основу исполняемых утверждений БЗ составляют правила и процедуры. Кроме того, есть формулы, функции, действия и т. п. Правила в G2 имеют традиционный вид [7]: условие (антецедент) и заключение (консеквент). Кроме *if*-правила условие («*if* <логическое выражение>») и заключение («*then* <действия>») используются еще 4 типа правил: *initially*, *unconditionally*, *when* и *whenever*.

Способы применения каждого правила определяются его синтаксисом [7]:

<правило>::= {| <префикс *for*>|
{<правило *if* >| <правило *unconditionally*>
| <правило *when*>| <правило *whenever*>}
| <правило *initially*>}
<префикс *for*>::= **for** {**any**| **the**} <*item*> …
<правило *if*>::= **if** <логическое выражение>
then <список действий>
<правило *unconditionally*>::= **unconditionally** <список действий>
<правило *when*>::= **when** <логическое выражение>
then <список действий>
<правило *whenever*>::= **whenever** <описание события>
| **or** <описание события>|
| **and when** <логическое выражение>|
<правило *initially*>::= **initially**
[**if** <логическое выражение> **then**] <список действий>

Каждый из типов правил может быть как общим, т. е. относящимся ко всему классу, так и *специализированным*, относящимся к конкретным экземплярам класса. Возможность представлять знания в виде общих, а не только конкретных, правил обеспечивает следующие преимущества:

- минимизируется избыточность БЗ;
- упрощается наполнение БЗ и ее сопровождение;
- минимизируются ошибки при отладке БЗ;
- способствует повторной используемости знаний (так как общие правила запоминаются в библиотеке G2 и могут использоваться в подобных приложениях).

Несмотря на то, что продукционные правила обеспечивают достаточную гибкость для описания реакций системы на изменения окружающего мира, в некоторых случаях, когда необходимо выполнить жесткую последовательность действий (например, запуск или остановку комплекса оборудования), более предпочтительным является процедурный подход. Язык программирования (ЯП), используемый в G2 для представления процедурных знаний, является достаточно близким родственником ЯП «Паскаль». Кроме стандартных управляющих конструкций язык расширен элементами, учитывающими работу процедуры в реальном времени: ожидание наступления событий, разрешение другим задачам прерывать выполнение данной процедуры, директивы, задающие последовательное или параллельное выполнение операторов. Еще одна интересная особенность языка — *итераторы*, позволяющие организовать цикл над множеством экземпляров класса. Перечисленные свойства языка позволяют системе одновременно выполнять множество различных процедур или множество копий одной и той же процедуры для множества различных объектов.

ЗАКЛЮЧЕНИЕ

В заключение обратим внимание на *ограничения* существующих ЭС и инструментариев в рассматриваемом аспекте состава и способа представления знаний.

Причинами ограничения состава знаний, используемых в ЭС, могут быть:

1) отсутствие знаний о причинных моделях, недостаточное представление знаний о разнообразии целей и возможностей пользователей, недостаточность знаний о собственных возможностях и ограничениях системы;
2) неполнота знаний, представленных в ЭС, следствием которой является ограничение возможных областей их использования;
3) отсутствие хорошо разработанных методов представления временных и пространственных знаний;
4) невозможность описания с помощью применяемых языков представления знаний всех отношений, которые эксперт считает важными, в едином формализме.

На основании вышесказанного в качестве одной из тенденций развития ЭС можно указать расширение сферы их использования за счет разработки более эффективных способов представления знаний.

СПИСОК БИБЛИОГРАФИЧЕСКИХ ССЫЛОК

1. Марселлус Д. Программирование экспертных систем на Турбо-Прологе : пер. с англ. / предисл. С. В. Трубицина. М. : Финансы и статистика, 1994. 256 с. : ил.
2. Рот М. Интеллектуальный автомат : компьютер в качестве эксперта : пер. с нем. М. : Энергоатомиздат, 1991. 80 с. : ил.
3. Экспертные системы на персональных компьютерах : материалы семинара. М. : МДНТП, 1990. 140 с.
4. Экспертные системы : принципы работы и примеры / А. Брукинг [и др.]; под ред. Р. Форсайта; пер. с англ. С. И. Рудаковой; под ред. В. Л. Стераника. М. : Радио и связь, 1987. 220 с. : ил.
5. Экспертные системы : сборник / ред. Б. М. Васильев. М. : Знание, 1990. 47 с. : ил.
6. Убейко В. М., Убейко В. В. Экспертные системы в СССР. Обзор информ. // Маш. пр-во. Сер. Автоматизированные системы проектирования и управления, вып. 5 (СНИИТЭМР). М., 1991. 67 с.

7. Статические и динамические экспертные системы : учеб. пособие для вузов / Э. В. Попов, И. Б. Фоминых, Е. Б. Кисель, М. Д. Шапот. М. : Финансы и статистика, 1996. 320 с. : ил.
8. Попов Э. В. Экспертные системы. Решение неформализованных задач в диалоге с ЭВМ. М. : Наука, 1987. 288 с. : ил.
9. Попов Э. В. Общение с ЭВМ на естественном языке. М. : Наука, 1982. 360 с.
10. Мендельсон Э. Введение в математическую логику. М. : Наука, 1971. 320 с.
11. Попов Э. В., Фридман Г. Р. Алгоритмические основы интеллектуальных роботов и искусственного интеллекта. М. : Наука, 1976. 455 с.
12. Мальцев А. А. Алгоритмы и рекурсивные функции. М. : Наука, 1965. 391 с.
13. Маслов С. Ю. Обратный метод установления выводимости в классическом исчислении предикатов // Доклады академии наук СССР. 1964. Т. 159, № 1. С. 17–20.
14. Ефимов Е. И. Решатели интеллектуальных задач. М. : Наука, 1982. 320 с.
15. Финн В. К. Индуктивные модели // Представление знаний в человеко-машинных и робототехнических системах. М. : ВИНИТИ, 1984. Т. А. С. 58–76.
16. Поспелов Д. А. Элементы аксиоматики временных отношений // Вопросы кибернетики. Ситуационное управление. Теория и практика. М. : Научный совет по комплексной проблеме кибернетики АН СССР, 1975. С. 15–21.
17. Поспелов Д. А. Логико-лингвистические модели в системах управления. М. : Энергоатомиздат, 1981. 231 с.
18. Поспелов Д. А. Псевдофизические логики // Представление знаний в человеко-машинных и робототехнических системах. М. : ВИНИТИ, 1984. Т. А. С. 48–57.

19. Поспелов Д. А. О «человеческих» рассуждениях в интеллектуальных системах // Вопросы кибернетики. Логика рассуждений и ее моделирование. М. : Научный совет по комплексной проблеме «Кибернетика», 1983. С. 5–37.
20. Поспелов Д. А. Логические модели представления знаний. Вводные замечания // Представление знаний в человеко-машинных и робототехнических системах. М. : ВИНИТИ, 1984. Т. А. С. 33–35.
21. Крипке С. А. Теорема полноты в модальной логике // Модальная логика. Фейс Р. М. : Наука, 1974. С. 223–246.
22. Крипке С. А. Семантический анализ модальной логики. Ч. I. Нормальные модальные исчисления высказываний // Модальная логика. Фейс Р. М. : Наука, 1974. С. 254–303.
23. Крипке С. А. Семантический анализ модальной логики. Ч. II. Нормальные модальные исчисления высказываний // Модальная логика. Фейс Р. М. : Наука, 1974. С. 304–323.
24. Гаврилова Т. А., Червинская К. Р. Извлечение и структурирование знаний для экспертных систем. М. : Радио и связь, 1992. 200 с. : ил.
25. The OPS_5 user's manual. Technical Rept. CMU-CS-81. Pittsburgh : Carnegie — Mellon University, 1981.
26. Walker C. T., Miller K. R. Expert system 1987 : An Assessment of Technology and Applications. Madison, 1988.
27. Кирсанов Б. С., Попов Э. В. Отечественные оболочки экспертных систем для больших ЭВМ // Искусственный интеллект : справочник. М. : Радио и связь, 1990. Т. 1. С. 369–388.
28. Хорошевский В. Ф. Программный инструментарий представления знаний в экспертных системах // Экс-

пертные системы : состояние и перспективы; под ред. Д. А. Поспелова. М. : Наука, 1989. С. 38–47.
29. Ковригин О. В., Перфильев К. Г. Гибридные средства представления знаний в системе СПИЭС // Всесоюзная конференция по искусственному интеллекту : тез. докл. Переславль-Залесский, 1988. Т. 2. С. 490–494.
30. Соловьев С. Ю., Соловьева Г. М. Вопросы организации баз знаний в системе ФИАКР // Экспертные системы : состояние и перспективы / под ред. Д. А. Поспелова. М. : Наука, 1989. С. 47–54.
31. Цейтин Г. С. Программирование на ассоциативных сетях // ЭВМ в проектировании и производстве. Л. : Машиностроение, 1985. Вып. 2. С. 16–48.
32. Сапатый П. С. Язык ВОЛНА-0 как основа навигационных структур для баз знаний на основе семантических сетей // Изв. АН СССР. Техническая кибернетика. 1986. № 5. С. 198–211.
33. Уварова Т. Г. Операционная семантика волновых языков и метод ее описания // Изв. АН СССР. Техническая кибернетика. 1987. № 2. С. 128–142.
34. Уотермен Д. Руководство по экспертным системам : пер. с англ. М. : Мир, 1989. 388 с.
35. Построение экспертных систем / под ред. Ф. Хейес-Рота, Д. Уотермена, Д. Лената. М. : Мир, 1987. 44 с.
36. Minsky M. A framework for representation knowledge // Psychology computer vision. New York : McGraw-Hill, 1975 (Русский перевод в кн. Психология машинного зрения. М. : Мир, 1978.)
37. Байдун В. В., Бунин А. И. Средства представления и обработки знаний в системе FRL/PS // Всесоюзная конф. по искусственному интеллекту : тез. докл. Переславль-Залесский, 1988. Т. 1. С. 66–71.
38. Шенк Р., Бирнбаум Л., Мей Дж. К интеграции семанти-

ки и прагматики // Новое в зарубежной лингвистике. Компьютерная лингвистика. М. : Прогресс, 1989. 432 с.
39. Bobrow D. G., Winograd T. KRL : another perspective // Cognitive Science. 1979. Vol. 3, № 1.
40. Duda R.0., Gaschnig J. G. Knowledge-based expert systems come of age // BYTE. 1981. V. 6, № 9. P. 238–281.
41. Zadeh L. A. Fuzzy sets // Information and Control. 1965. V. 8. P. 338–353.
42. Duda R. O., Hart P. E., Nlsson N. J. Subjective Bayesian methods for rule-based inference systems // Proceedings of the AFIPSS, National Computer Conference. 1976. V. 45. P. 1075–1082.
43. Greiner R., Lenat D. A representation language // The first National Conference on Artificial Intelligence. Stanford : Stanford University Press, 1980. P. 165–169.
44. Erman L. D., London P., Fickas S. F. The design and an example use of HEARSAY-III // The Seventh International Joint Conference on Artificial Intelligence. Vancouver : University of British Columbia, 1981. P. 409–415.
45. Алексеева Е. Ф., Стефанюк В. Л. Экспертные системы (состояние и перспектива) // Известия АН СССР. Техническая кибернетика. 1984. № 5. С. 153–167.
46. Ковригин О. В., Смольянинов Н. Д., Чмырь А. Я. Экспертные медицинские диагностирующие системы // Известия АН СССР. Техническая кибернетика. 1982. № 5. С. 199–216.
47. Клещев А. С., Черняховская М. Ю. Медицинские системы-консультанты // Представление знаний в человекомашинных и робототехнических системах. М. : ВИНИТИ, 1984. Т. С. С. 282–309.
48. Хорошевский В. Ф. Инструментальные экспертные системы // Представление знаний в человеко-машинных и робототехнических системах. М. : ВИНИТИ, 1984. Т. С. С. 329–367.

49. Buchanan B. G., Sutherland G. L., Feigenbaum E. A. Heuristic DENDRAL : A program for generating explanatory hypotheses in organic chemistry // Machine intelligence. Edinburgh : Edinburgh University Press, 1969. Vol. 4. P. 209–254.
50. Feigenbaum E. A., Buchanan B. G., Lederberg J. On generality and problem solving : A case study using the DENDRAL program // Machine intelligence. Edinburgh : Edinburgh University Press, 1971. Vol. 6. P. 209–254.
51. Shortliffe E. H. Computer-based medical consultation : MYCIN. New York : Academician Elsevier, 1976. 264 p.
52. Davis R. Interactive Transfer of expertise : Acquisition of inference rules // 5th International Joint Conference on Artificial Intelligence. Boston : MIT, 1977. P. 321–328.
53. Davis R. Knowledge acquisition in rule-based systems : Knowledge about representation as a basic for system construction and maintenance // Pattern directed inference systems. New York : Academic Press, 1978. P. 99–134.
54. Davis R. TEIRESIAS : Applications of meta-level knowledge // Knowledge-based system in artificial intelligence. New York : McGraw-Hill, 1982. P. 229–484.
55. Van Melle W. A domain-independent production-rule system for consultation programs // The Sixth International Joint Conference on Artificial Intelligence. Tokyo, August 1979. P. 1149.
56. Duda R. O., Gaschnig J. G., Hart P. E. Model design in PROSPECTOR consultant system for mineral exploration // Expert system in the micro-electronic age. Edinburgh : Edinburgh University Press, 1979. P. 153–167.
57. Хендрикс (Hendrix G. G.). О расширении применяемости семантических сетей введением разбиений // Труды IV Международной объединенной конференции по искусственному интеллекту. Тбилиси, 1975. Т. 1. С. 190–206.

58. Pople H. E. The formation of composite hypotheses in diagnostic problem solving : An exercise in synthetic reasoning // The fifth International Joint Conference on Artificial Intelligence. Boston : MIT, 1977. P. 1030–1037.
59. McDermott J. R1 : An expert in the computer system domain // The first National Conference of Artificial Intelligence. Stanford : Stanford University Press, 1980. P. 269–271.
60. Nilsson N. J. Principles of Artificial Intelligence. Palo Alto, California : Tioga Press, 1980. (В 1985 г. русский перевод книги вышел в издательстве «Радио и связь»).
61. Fikes R. E., Nilsson N. J. STRIPS: A New Approach to the Application of Theorem Proving to Problem Solving // Artificial Intelligence. 1971. № 2. P. 189–208.
62. Forgy C. The OPS 5 user's manual. Technical Report. CMU-CS-81–135, Computer Science Department, Carnegie-Mellon University. Pittsburgh, Pa., 1981.
63. Forgy C., McDermott J. OPS : A domian-independent production system language // The fifth International Joint Conference on Artificial Intelligence. Boston : MIT, 1977. P. 933–939.
64. Forgy C. RETE : A fast algorithm for the many pattern / many object pattern match problem // Artificial Intelligence. 1982. V. 19. P. 17–38.
65. The ROSIE language reference manual. Tech. Report, N-1647-ARPA, Rand. Corporation. / J. D. Fain, F. Gorlin, F. Hayes-Roth, S. J. Kosenschein. Santa Monica, California, 1981. P. 38.
66. Waterman D. A. User-oriented systems for capturing expertise : a rule-based approach // Expert system in the microelectronic age. Edinburgh : Edinburgh University Press, 1979. P. 26–34.
67. HEARSAY-III : A domian-independent framework for expert system / R. Balser, L. D. Erman, P. London, C. Wil-

liams // The first National Conference on Artificial Intelligence. Stanford : Stanford University Press, 1980. P. 108–110.
68. Lesser V. R., Erman L. D. A retrospective view of the HEARSAY-III // The fifth International Joint Conference on Artificial Intelligence. Boston : MIT, 1977. P. 790–800.
69. Попов Э.Я, Кирсанов Б. С. Инструментальный комплекс для построения на ПЭВМ эффективных экспертных систем в широком классе приложений // Материалы Второй всесоюзной конференции по искусственному интеллекту. Минск, 1990. Т. 3. С. 12–14.
70. Попов Э. В. Экспертные системы реального времени // Открытые системы. 1995. № 2. С. 66–71.
71. Попов Э. В., Фоминых И. Б., Кисель Е. Б. Статические и динамические экспертные системы (классификация, состояние, тенденции) : методические материалы. М. : ЦРДЗ, 1995. 157 с.

ОГЛАВЛЕНИЕ

Учебное издание

ПРЕДСТАВЛЕНИЕ ЗНАНИЙ В ЭКСПЕРТНЫХ СИСТЕМАХ

Составители **Морозова** Вера Анатольевна
Паутов Валентин Иванович

Редактор *Н. П. Кубыщенко*
Компьютерный набор *В. А. Морозовой*
Верстка *Е. В. Ровнушкиной*

Подписано в печать 03.04.2017. Формат 60×84 1/16.
Бумага писчая. Цифровая печать. Усл. печ. л. 7,0.
Уч.-изд. л. 5,2. Тираж 50 экз. Заказ 71.

Издательство Уральского университета
Редакционно-издательский отдел ИПЦ УрФУ
620049, Екатеринбург, ул. С. Ковалевской, 5
Тел.: 8 (343) 375-48-25, 375-46-85, 374-19-41
E-mail: rio@urfu.ru

Отпечатано в Издательско-полиграфическом центре УрФУ
620075, Екатеринбург, ул. Тургенева, 4
Тел.: 8 (343) 350-56-64, 350-90-13
Факс: 8 (343) 358-93-06
E-mail: press-urfu@mail.ru

МОРОЗОВА ВЕРА АНАТОЛЬЕВНА
Доцент Департамента информационных технологий и автоматики, кандидат технических наук, доцент Института радиоэлектроники и информационных технологий – РтФ Уральского федерального университета.
Область научных интересов: экспертные системы управления; принятие решений; моделирование представления знаний в экспертных системах управления.

ПАУТОВ ВАЛЕНТИН ИВАНОВИЧ
Доцент Департамента информационных технологий и автоматики, кандидат технических наук, доцент Института радиоэлектроники и информационных технологий – РтФ Уральского федерального университета.
Область научных интересов: аппаратные инструментальные средства разработки экспертных систем; интеллектуальные электронные датчики.